Universidade de Aveiro Departamento de Matemática
2017

Joana Patrícia Bordonhos Ribeiro

Identificação de *Outliers* em Séries Temporais Multivariadas

Outlier Identification in Multivariate Time Series

Joana Patrícia Bordonhos Ribeiro

Identificação de *Outliers* em Séries Temporais Multivariadas

Outlier Identification in Multivariate Time Series

"*If you torture the data long enough, it will confess.*"

— Ronald Coase

Universidade de Aveiro Departamento de Matemática
2017

Joana Patrícia Bordonhos Ribeiro

Identificação de *Outliers* em Séries Temporais Multivariadas

Outlier Identification in Multivariate Time Series

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Matemática e Aplicações, realizada sob a orientação científica do Doutor Luís Miguel Almeida da Silva (orientador), Professor auxiliar convidado do Departamento de Matemática da Universidade de Aveiro.

o júri / the jury

presidente / president

Prof. Doutor Pedro Filipe Pessoa Macedo
Professor Auxiliar da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor Jorge Manuel Fernandes dos Santos
Professor Adjunto do Departamento de Matemática do Inst. Superior de Engenharia do Porto, do Inst. Politécnico do Porto

Prof. Doutor Luís Miguel Almeida da Silva
Professor Auxiliar Convidado da Universidade de Aveiro

agradecimentos /
acknowledgements

Agradeço todo o apoio prestado pelos meus pais ao longo deste trabalho e a motivação dada para constantemente procurar melhorar.
Agradeço também a todos os meus colegas da Bosch, nomeadamente à minha orientadora Tânia Correia, que desde o primeiro dia me fizeram sentir integrada e me motivaram a procurar soluções para todos os problemas enfrentados ao longo deste percurso.
Um agradecimento ao meu orientador, professor Luís Silva, pela disponibilidade prestada e pela exigência pedida que permitiu a conclusão deste trabalho na forma em que se apresenta.
Por último, um agradecimento ao Fábio, elemento especial da equipa Bosch, por me motivar a alcançar o melhor de mim.

Palavras Chave Erros, Séries Temporais, Classificação, Multivariado.

Resumo

Com o desenvolvimento tecnológico, existe uma cada vez maior disponibilidade de dados. Geralmente representativos de situações do dia-a-dia, a existência de grandes quantidades de informação tem o seu interesse quando permite a extração de valor para o mercado. Além disso, surge importância em analisar não só os valores disponíveis mas também a sua associação com o tempo.

A existência de valores anormais é inevitável. Geralmente denotados como *outliers*, a procura por estes valores é realizada comummente com o intuito de fazer a sua exclusão do estudo. No entanto, os *outliers* representam muitas vezes um objetivo de estudo. Por exemplo, no caso de deteção de fraudes bancárias ou no diagnóstico de doenças, o objetivo central é identificar situações anormais.

Ao longo desta dissertação é apresentada uma metodologia que permite detetar *outliers* em séries temporais multivariadas, após aplicação de métodos de classificação. A abordagem escolhida é depois aplicada a um conjunto de dados real, representativo do funcionamento de caldeiras. O principal objetivo é identificar as suas falhas. Consequentemente, pretende-se melhorar os componentes do equipamento e portanto diminuir as suas falhas.

Os algoritmos implementados permitem identificar não só as falhas do aparelho mas também o seu funcionamento normal. Pretende-se que as metodologias escolhidas sejam também aplicadas nos aparelhos futuros, permitindo melhorar a identificação em tempo real das falhas.

Keywords Outliers, Time Series, Classification, Multivariate

Abstract

With the technological development, there is an increasing availability of data. Usually representative of day-to-day actions, the existence of large amounts of information has its own interest when it allows to extract value to the market. In addition, it is important to analyze not only the available values but also their association with time.

The existence of abnormal values is inevitable. Usually denoted as *outliers*, the search for these values is commonly made in order to exclude them from the study. However, outliers often represent a goal of study. For example, in the case of bank fraud detection or disease diagnosis, the central objective is to identify the abnormal situations.

Throughout this dissertation we present a methodology that allows the detection of outliers in multivariate time series, after application of classification methods. The chosen approach is then applied to a real data set, representative of boiler operation. The main goal is to identify faults. It is intended to improve boiler components and, hence, reduce the faults.

The implemented algorithms allow to identify not only the boiler faults but also their normal operation cycles. We aim that the chosen methodologies will also be applied in future devices, allowing to improve real-time fault identification.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The world is one big data problem. - Andrew McAfee

A constant search for innovation allowed an increasing capacity to collect the most varied types of information from real-life events. With this, a greater difficulty in constructing machine learning algorithms capable of dealing with new data interest aspects, such as changeable behaviors, also appeared. Data sets are usually representative of day-to-day actions. Therefore, machine learning algorithms have to focus not only on the variable values, but also in their relation with time. Instead of unchangeable variables, considered in the most frequent case studies such as petal length and width in the Iris flower data set, data is now seen as a vast and complex set of variable evolutions representative of changing behaviors through time. Besides that, data has a meaning as a whole, that is, as a set of variables that can be seen separately but in which nothing can be concluded by observing each variable by itself. Instead, the relation between them has to be considered. Limitations of memory and processing time due to increasing data starts to be a new difficulty for implementing machine learning algorithms.

When collecting data from real world problems, abnormal values sometimes appear. Usually denoted as *outliers*, they are different from what is considered to be normal in each specific case study. In some situations, these values are not common data set noise but, instead, they represent changes or malfunctions in the systems [1]–[6]. Fraud detection in bank accounts, search for existing cancer cells or environmental changes detection are some examples where it is possible to understand that outliers can have their own interest. In this sense, several methods have been developed to allow outlier identification, enabling a more effective human intervention. For example, in bank fraud detection it would be difficult to

analyze each account movement in order to find anomalies. However, with outliers being identified by some machine learning algorithm, bank workers can analyze previous customer habits in order to individually detect if that specific movement appears to be unusual or not. After that, one possible step is to approach the customer and make a decision about the identified fraud. Therefore, machine learning algorithms capable of identifying outliers could provide a quicker human intervention.

The data in this work is representative of actual behaviors from boilers, which may not always properly perform their essential function of hot water supply. However, since they are running daily in customers' houses, normal operation is more frequent than abnormal events, implying a low frequency of outliers in the data set. This usually translates into problems for the machine learning traditional methods. An unbalanced data set commonly results in a classification preference for the most frequent class, mainly because most machine learning methods are statistically based. Some techniques exist trying to overcome this fact, such as undersampling or oversampling. In the former, majority class observations are randomly picked in the same number as the minority class, although relevant information from the majority class may be lost; in the latter, the minority class observations are duplicated to match the majority class size, although overfitting phenomena may occur, due to the repeated samples of the minority class[1].

Studied appliance malfunctions are associated with faults identified by the appliance software and automatically tagged with a fault code. Thus, we have labeled data, that is, there exists a target variable corresponding to the fault code, but also data concerning appliance malfunctions that were not identified, resulting in unlabeled observations. An exhaustive study of boilers operation in order to understand why the faults occur allowed to associate some behavior patterns with each fault. Although the results are not very enlightening or reliable, it is not also an easy task to enable a machine to follow the human logic that leads to fault identification. Moreover, it would be impractical to manually label each of those unidentified faults. So, the necessity for algorithms capable of learning patterns with the goal of labeling data, naturally appeared. The final result of this work is to identify boiler malfunctions through time series outlier identification techniques [7]–[9].

Two methods were considered to solve the present problem: novelty detection [10] and value-trend approach [11]. The novelty detection family of methods tries to find different values from what is considered to be a normal behavior. Generally, it consists in obtaining a normal operation model and then, attribute the fault label to all behaviors that do not fit in the model, considering some threshold. This approach was not applied due to time restrictions. The value-trend approach mainly consists in representing time series by strings, reducing

[1]Briefly, overfitting corresponds to a model that fits the training set too well, capturing the undesirable details and noises, while underfitting is the opposite, that is, when the model is not capable of capturing the relationship between predictive and response variables, presenting a poor performance.

variables possible values. Simultaneously, since sets of consecutive values are transformed into one single letter, the needed memory also decreases. This data transformation allows to apply some common machine learning algorithms and solve the time series classification problem. So, in the value-trend approach, fault codes and normal operation cycles are considered as labels. The value-trend approach was the solution applied in the presented work.

The present data is from boilers produced by the Thermotechnology division of Bosch, world leader in development and production of heating and hot water systems with high energy efficiency. In Portugal, this division is represented by the Bosch Aveiro factory - original Vulcano - with 40 years of existence. There, boilers, heat pumps and water heaters are produced and exported to more than 30 countries worldwide. In the Bosch Aveiro I&D building, the first compact thermostatic water heater with connectivity technology and remote control through a smartphone or tablet was developed. In the company context, boiler fault identification that occurred in the past but also the ones that continue to occur in customer's houses, allows the improvement of components and the maintenance in advance of already functioning appliances. By identifying the most common faults, it is possible to substitute in future models the components with higher failure rates. Also, a more extensive and accurate fault identification will allow a quicker and effective correction of boiler malfunctions by the warranty maintenance responsible, since each fault code corresponds to a specific component failure being documented with a possible solution. It is also intended to decide the lifetime of each model by considering the fault occurrences over time.

In Chapter 2, data characteristics will be presented in order to understand the proposed problem and introduce it into machine learning families. Also, the choice of variables and time range considered to identify the labels is made. The value-trend approach and the classification methods able to solve the under study problem are present in Chapter 3. The steps of data processing, algorithm implementation and comparison of performance measures are discussed in Chapter 4. In the end, a short discussion of possible future improvements is made.

Chapter 2

The Data

Somewhere in me is a curiosity sensor. I want to know what's over the next hill. You know, people can live longer without food than without information. Without information, you'd go crazy. - Arthur C. Clarke

The first step in any machine learning process is to know the system under study and consequently the characteristics of the data. Only by this way it is possible to consciously choose methods able to deal with the problem at hand and obtain a correct interpretation of the results. Usually, a final output translation into the original problem language is necessary to allow a general understanding of the results by all. So, the complete knowledge of the data in study is a crucial point to solve the problem.

The proposed data is obtained from several different boilers operating in customers' houses and produced by Bosch. These appliances main function is to supply hot water that can be requested to use as domestic water, such as hot water for bathing or kitchen, or as central heating, in which hot water closed circulation along the house allows to heat wall-hung heating devices. In this way, there are two main cycles: domestic hot water - DHW - and central heating - CH. There is an additional important mode called boost cycle, which provides a quicker supply of hot water. These cycles are considered normal operations and although they are the most frequent in the data, they are not the goal of this study. Nevertheless, one approach to outlier identification may be to learn normal operation cycles and then labeling as faults the behaviors that do not fit in such patterns. So, it is also important to consider boilers normal operation in order to identify which behaviors are not considered normal and, consequently, faults.

The appliances in study have sensors collecting data concerning, for example, temperatures, flow and number of heating requests made, but also state variables such as open or close valves,

requests for heating, among others. Therefore, the data set is constituted by continuous and discrete variables. Note that state variables, originally qualitative, can be transformed in discrete values such as binary. Besides that, every such variable always assumes two opposite values like On/Off, Yes/No, HW/CH, among others. So, for example the states "On" and "Off" concerning the "Central Heating Request" variable can be coded as 1 and 0, respectively. Thus, the data set is originally multivariate with quantitative and qualitative variables that were transformed just into quantitative ones.

The data set is constituted by 40 discrete and 29 continuous variables. There are 4 additional variables concerning the appliance model and the connectivity gateway software version. The connectivity gateway is an important device installed in the studied boilers that allows data transmission. This two variables vary for each boiler but do not change over time. However, they have to be considered because some faults are related with specific appliance models or gateway software versions. These are string type variables that were not transformed into quantitative ones. One last, very important variable, concerns the fault code. This identification is automatically made by the appliance software in real-time with the goal of allowing a faster and more efficient defective components correction or substitution by the maintenance responsible in the customer's house. Remember that this work main goal is the construction of an algorithm able of better identifying faults, since it is known that some were not automatically identified. Therefore, fault occurrences associated with one fault code are considered labeled observations that will be used to construct the model. In this way, the outlier identification initial task becomes a time series classification problem.

The data was collected within one year and four months concerning 1563 appliances of about 5 different models. Although data is received every millisecond, only the variables for which there has been any change in the value or state are recorded. This will be an important detail in data processing, later in Chapter 4.

Consider the data matrix reduced example of Figure 2.1 concerning three variables during 60 milliseconds, where it is possible to understand this recording process which occurs only when there are changes (although the appliance is always recording). Considering, for example, the variable "Water Temperature", the value 20 has been recorded at $01:30:00.000$. Since there were no changes for this variable in the next milliseconds, the matrix entries are empty (lines $2, 3$ and 4 of the second column). However, at $01:30:00.400$ a new value has been detected: 22. So, a new entry was recorded in the line corresponding to the time of the change and in the "Water Temp." column.

Through data visualization of normal operation cycles and identified faults, it was possible to exclude variables with no labeling influence, such as those with no value variations from label to label. There are also variables with no saved values since some specific sensors only exist in appliances not considered in this study or just for a part of the

$$\mathbf{TS} = \begin{bmatrix} Time & Water\ Temp. & Gas & Fan \\ 01:30:00.000 & 20 & & \\ 01:30:00.100 & & 1 & \\ 01:30:00.200 & & & 1 \\ 01:30:00.300 & & & 0 \\ 01:30:00.400 & 22 & & \\ 01:30:00.500 & 23 & 0 & \end{bmatrix}$$

Figure 2.1: Data recording example.

studied models. It was considered that these variables do not characterize the behaviors of interest and so they were excluded from the set resulting in a reduction from 74 to 40 variables. Later, a statistical study of means and percentage of missing values that can be observed in Figure 2.2 allowed to exclude 9 more variables in the following way: 3 variables that presented more than 80% of missing values; 5 binary variables with mean value above 0.97 or below 0.03; 1 continuous variable with mean value above 97. So, in the end there were 31 variables instead of the original 74, where 4 are string variables, 16 assume binary values, 10 are continuous variables and one last variable concerns the fault code.

The variables originally considered assume values from 0 to 153 with a standard deviation varying between 0 and 53.6, being some of them binary variables. Standardization techniques such as the Z-score or the Min / Max Normalization were not applied in this phase. In addition, the hypothesis of reducing data dimensionality through, for example, Principal Component Analysis, was avoided since it is important to maintain the original data interpretability until the end in order to allow future critical analysis of boiler components.

The appliances software is able to identify 39 different faults, attributing to each one a distinct code. Each fault code is related with a specific boiler component failure and possible maintenance solutions are documented. Among others there are, for example, failures in the pump operation not allowing a correct water circulation, failures in the fan or in the gas valve opening or closing. Besides that, data loss due to connection problems or sensor malfunctions are also associated with a fault code. Thus, a total of 42 classes were considered for the time series classification problem, corresponding to the 3 previously discussed normal operation cycles and all the 39 existing fault codes.

Labeled occurrences per fault vary between 1 to 133 in a total of 1185 classified cycles. As is possible to see in Figure 2.3, this is clearly an unbalanced data set that can influence the learning process resulting in a classification preference for the majority class. However, this is also a reflection of more or less incident faults. For example, a fault that occurs frequently, and so with more labeled occurrences, is a fault that is expected to happen with high frequency in the test set. Therefore, methods to overcome this problem will not be considered in this study.

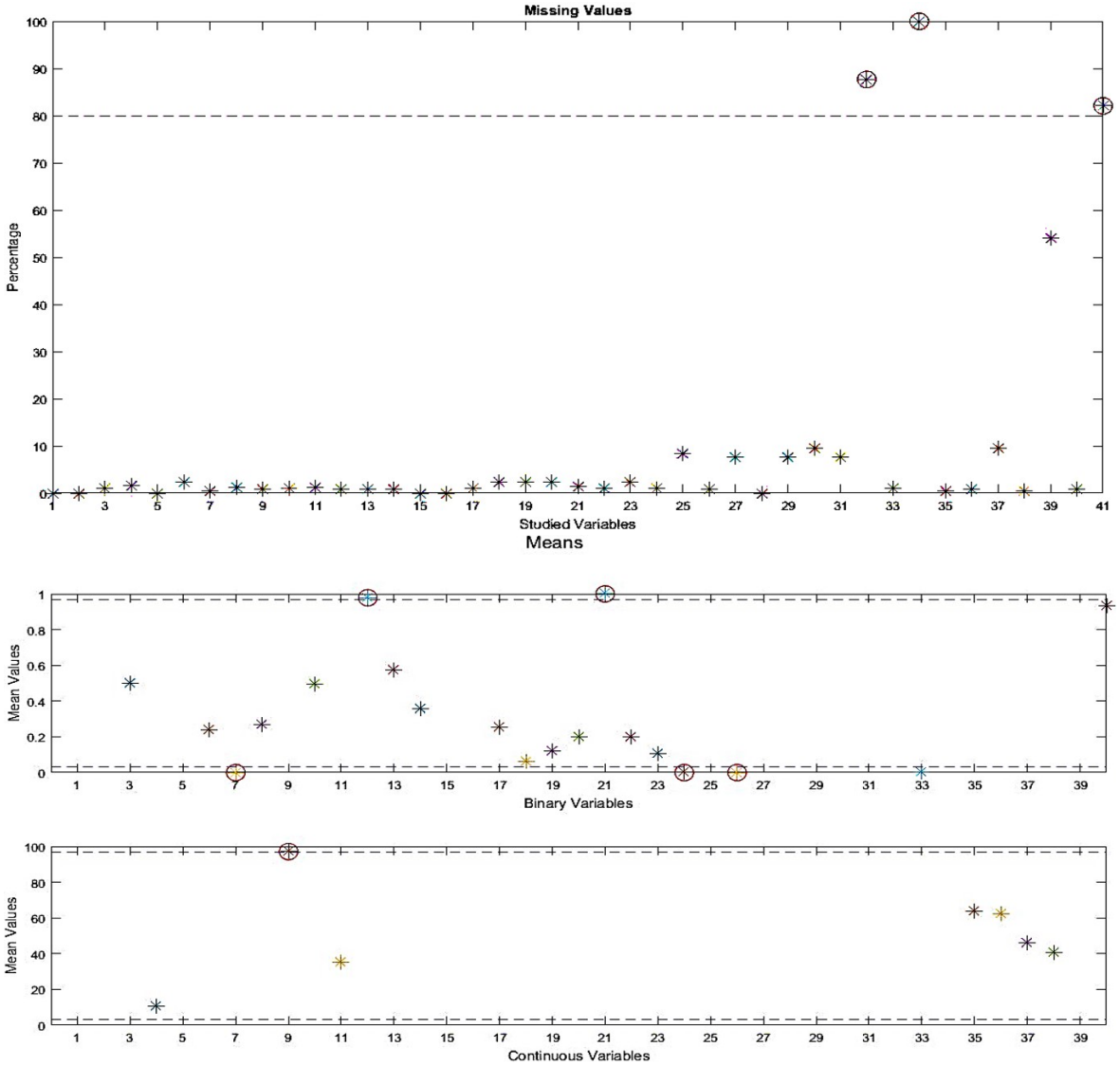

Figure 2.2: Statistical analysis of the percentage of missing values and mean value of each predictive variable. Horizontal lines represent threshold exclusion values.

After the initial analysis of some faults and normal operation cycles, a total time span of 45 minutes was considered, concerning 30 minutes before each label identification until 15 minutes after. Note that the data is collected from the MONGO database[1] where data concerning several days and months for each appliance operation is available.

Some faults are frequently identified several times in a short time period. Thus, with the previous time range criteria, this could lead to data repetition resulting in a model learning process with repeated data. For example, consider Figure 2.4 that shows a time series where the same fault occurred 7 times in about 4 minutes. As one can see, 30 minutes before the second fault occurrence and 15 minutes after the first correspond to the same 15 minutes of

[1] `https://www.mongodb.com/`

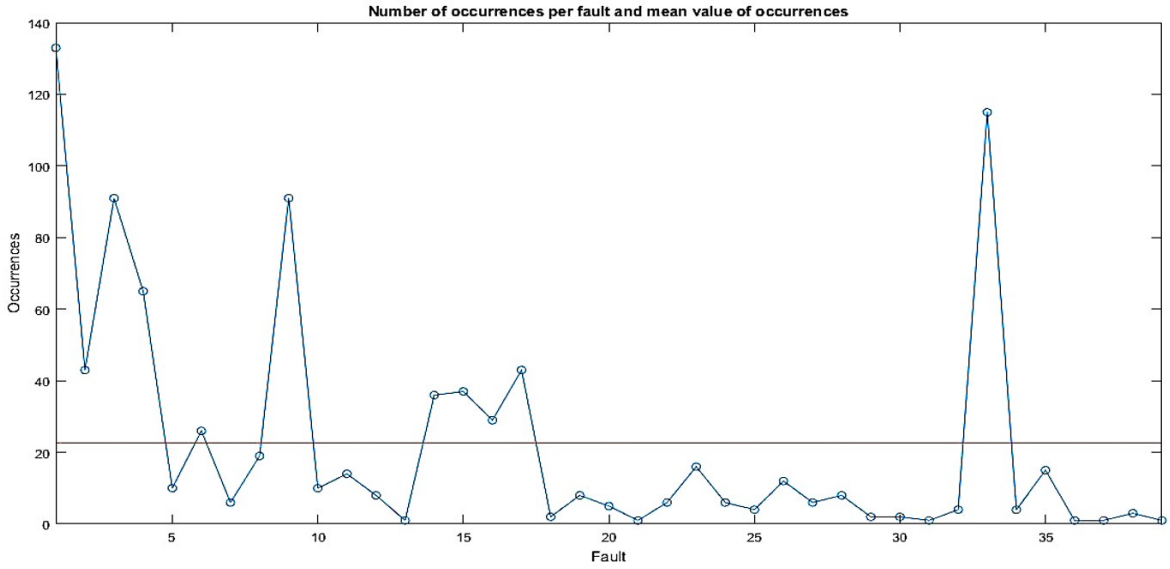

Figure 2.3: Number of occurrences per fault and mean value of occurrences (horizontal line).

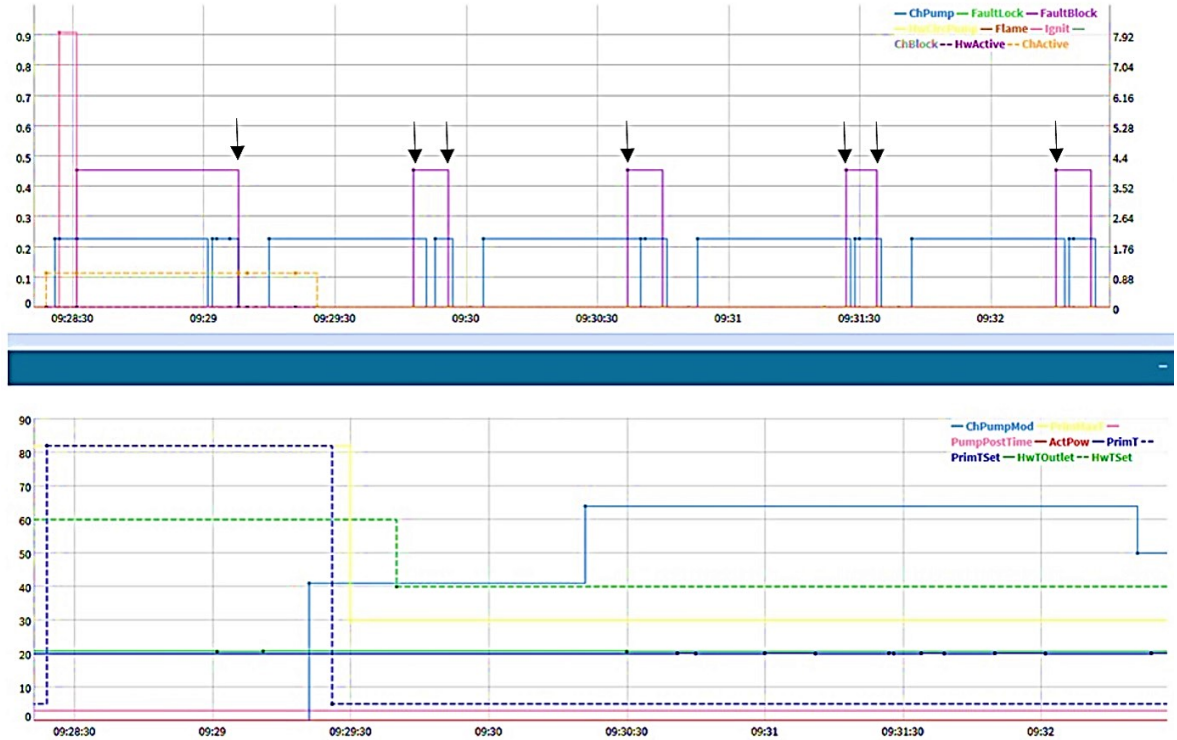

Figure 2.4: Time series with the same fault occurring 7 times in about 4 minutes.

data. So, in the learning process the same data would be considered twice, since the faults time difference is of about 30 seconds. Therefore, in similar situations, if the time range of the same label intersects, then the data is collected 30 minutes before the first fault occurrence and 15 minutes after the last one. This correction simply traduces in excluding equal lines of the data matrix considering equal labels, dates and times, after collecting the initial time range of data, as will be discussed in more detail in Chapter 4.

There are three exclusive behaviors present in the data that should be pointed out, although they are not labeled as faults.

First, data loss due to connection problems is a distinct fault from the absence of values related to sensors faults. However, they are both characterized by data matrix empty entries. In this specific case study, value absence for all collected variables for more than 4 minutes is considered data loss. An exhaustive data study has shown that state variables transmit values about every 4 minutes, even if there is not any value change for that variable.

Second, value absence reflected in empty matrix entries is different from assuming that a given variable has value zero. Considering for example temperature variables, a zero value is different from a value absence (for example, due to data loss or temperature sensors fault). Obviously, in this case the zero value has a clear meaning. Furthermore, since data is received within few seconds, it is assumed that the variable last value has not suffered changes until a new value is recorded for this specific variable. That is, continuity is adopted until a new value happens.

The third issue is related to this specific data collection process due to the connectivity system. Sometimes a value change in one variable happens before a second change. However, it is common to have connection problems that lead to the first change being received after the second one. This looks harmless. However, due to the enormous relationship between the variables over time not only in each variable independently, but also between them, this aspect could translate into a misinterpretation. Thus, it could lead to a fault identification, assuming some appliance fault, when in fact it is a common connectivity fault and so there was no problem with the boilers physical components. By now, this is not a correctable fault since it is impossible to identify with the available variables, and therefore it is not the aim of this study.

Chapter 3

Theoretical Background

(...) we need to understand that lack of data is not the issue. Most businesses have more than enough data to use constructively; we just don't know how to use it. - Bernard Marr

The main difficulty of the proposed problem is that the behaviors are represented by multivariate time series with several specificities that common machine learning algorithms have difficulty to deal with. Some variables are represented in the format of strings, like the boiler name, and others as numerical, such as temperatures. Also, there is a big importance in capturing the evolution of each variable, in order to understand the behaviors, and relate all the variables. Several algorithms have been proposed to deal with time series data [12]–[19] but we could not found any able to solve our specific problem. The algorithm studied in this section is responsible for performing a representation of the information present in a time series in a simpler form, enabling an easier implementation of common machine learning algorithms.

Briefly, the chosen algorithm has the purpose of compactly represent the variables evolution of each time series, also performing a reduction of dimensionality without a high loss of information [11], [20]. This representation is made through the construction of vectors representing the region of values assumed by the variable and its tendency in each time segment of the time series. After the construction of the new representation, usual techniques of machine learning that are able to solve the problem of supervised classification under study were applied. This technique was devised by Eamonn Keogh and Jessica Lin in 2002, consisting of two main steps: the transformation of the time series into sets of vectors through the piecewise aggregate approximation algorithm (PAA) and the conversion of those vectors into a set of letters by the symbolic aggregate approximation algorithm (SAX) [20], [21]. An additional phase where a trend analysis is performed as an improvement of the SAX

algorithm is also discussed.

3.1 Piecewise Aggregate Approximation

The first step of the transformation consists in reducing the time series dimension. With this purpose, the piecewise aggregate approximation (PAA) algorithm is used to divide the time series into segments [22], [23]. By definition, this algorithm transforms any time series X of length m into n segments of time, resulting in a vector $X = (x_1, x_2..., x_n)$ of temporal segments, where n is any arbitrary integer such that $n \leq m$. Therefore, the PAA algorithm allows the transformation of the time series into a vector of equally sized segments. After that, considering each of the segments constructed in the previous step, the average value of each variable is calculated. So, the PAA algorithm receives as input a set of data relative to a time series and divides it into segments. Then, for each temporal segment i, with $i = \{1, 2, ..., n\}$, and each variable j, the mean value w_{ji} is calculated.

Consider Figure 3.1 where the PAA algorithm is applied to a univariate time series. In the first step, the time series is divided into seven equally spaced temporal segments, represented by the dashed lines. Then, the mean value of the time series in each segment is calculated. These mean values are represented in Figure 3.1 by the orange lines. Since we are considering a univariate time series, the final result of the PAA algorithm is a single vector with seven mean values. Thus, the original time series is represented by a vector of mean values. If we

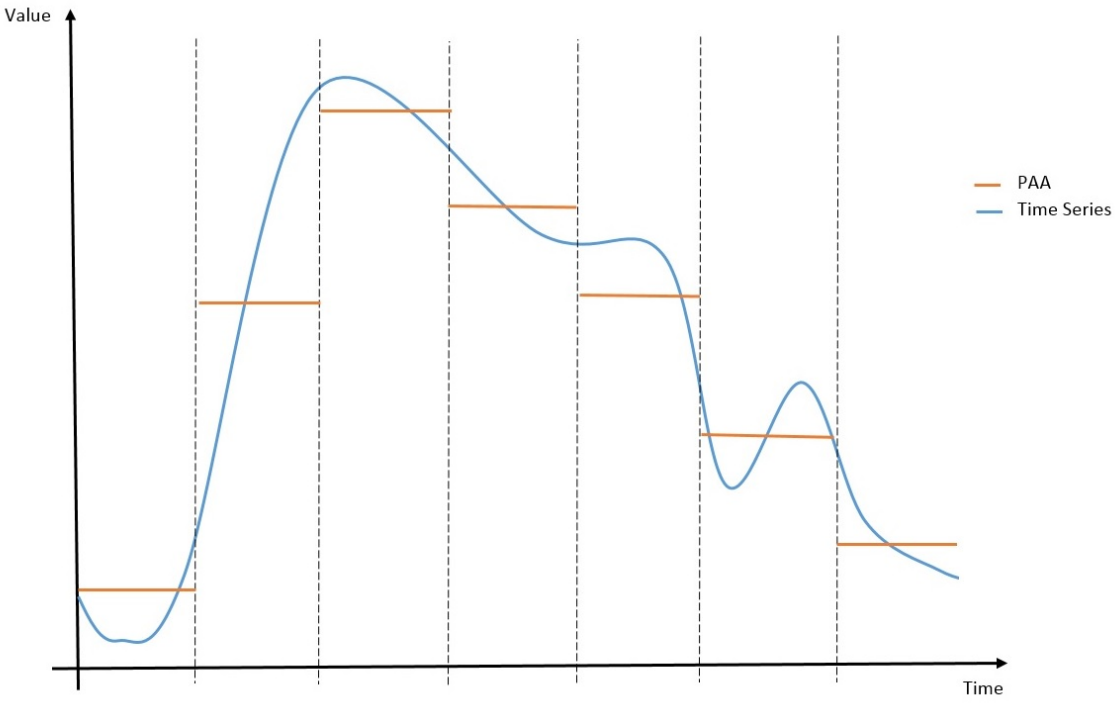

Figure 3.1: Application of the PAA algorithm in a univariate time series.

$$\mathbf{TS} = \begin{bmatrix} x_{11} & x_{12} & ... & x_{1k} \\ x_{21} & x_{22} & ... & x_{2k} \\ ... & ... & ... & ... \\ x_{m1} & x_{m2} & ... & x_{mk} \end{bmatrix} \Rightarrow \mathbf{PAA} = \begin{bmatrix} < w_{11}, w_{12}, ..., w_{1n} > \\ < w_{21}, w_{22}, ..., w_{2n} > \\ ... \\ < w_{k1}, w_{k2}, ..., w_{kn} > \end{bmatrix}^{\top}$$

Figure 3.2: Application of the PAA algorithm in matrix format.

now apply the PAA algorithm to a bidimensional time series and consider four segments, the final result would be the representation of the whole time series as two vectors each of which with four entries, stored as a matrix.

Note that the number of segments to be considered can vary between only one segment, and in this case the entire time series is equal to as many univariate vectors as the number of variables under study, up to the number of value registrations considered for the time series, taking each vector the same dimension as the time series. So, the number of segments is chosen by the user but this choice should be made with some caution.

In the specific case of the data set under study, after applying the PAA algorithm, we will have the same number of vectors as the number of considered variables. Each vector will have dimension equal to the number of segments considered to represent each of the time series. Thus, each time series is represented by a set of multidimensional vectors, all with the same dimension. Besides that, all time series will have the same number of vectors since the number of variables considered for the study is equal independently of the label. The dimension of the vectors could not always be the same from time series to time series, since the cycles have quite different lengths, from seconds to several minutes. The number of segments was calculated with respect to the time series length with the objective of obtaining representations with about the same temporal space.

Consider the time series present in the left of Figure 3.2 represented as a matrix of m lines and k columns, corresponding to m time registrations of k variables. After the application of the PAA algorithm we would obtain the matrix in the right side of the same Figure.

Note that we have been considering the partition of the time series into n segments and, so, we already have a dimension reduction. Each w_{ji} represents the mean value of the j-th variable in the i-th segment, with $j = \{1, 2, ..., k\}$ and $i = \{1, 2, ..., n\}$.

3.2 Symbolic Aggregate Approximation

The main process of the second step, the SAX algorithm, consists in attributing a letter for each mean value previously obtained with the PAA algorithm. An extensive and rigorous analysis performed in [24] has shown that time series data, after being normalized by Z-score,

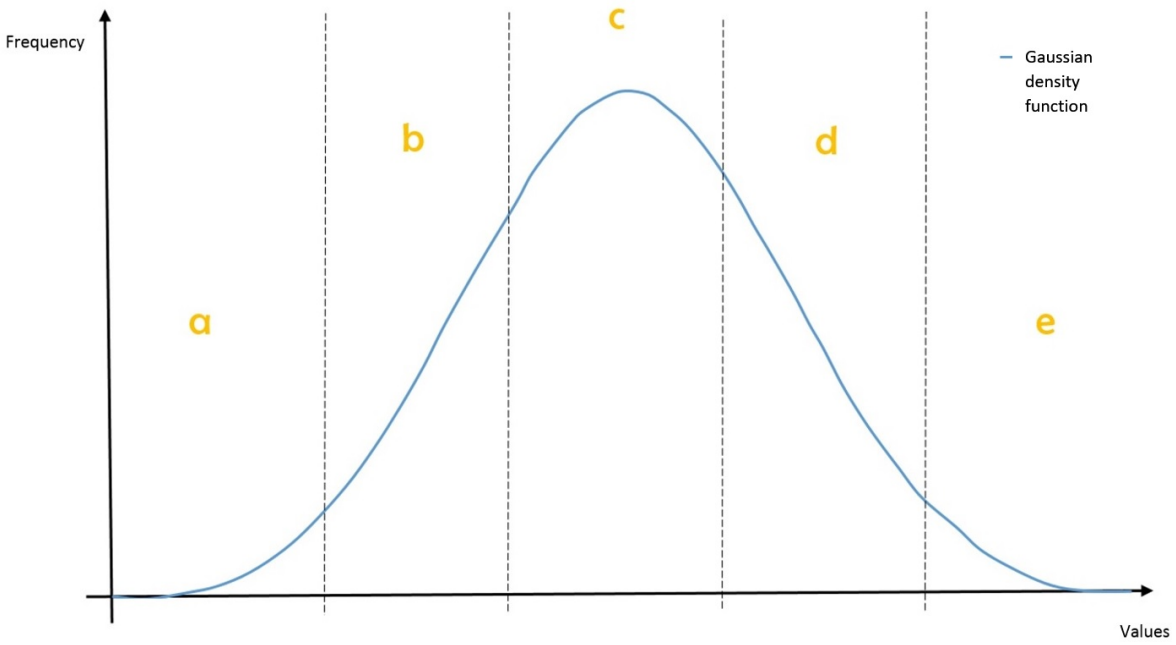

Figure 3.3: Attribution of letters based on the Gaussian distribution.

usually follows a Gaussian distribution. This detail enables a conscious attribution of letters to the mean values. Therefore, the partition of the area under the Gaussian probability density function into equally spaced breakpoints is possible. This partition must be performed according to the number of letters that we want to associate with the mean values of the vectors previously constructed.

In a succinct way, the SAX algorithm associates a letter to each area obtained by partitioning the probability density function (see Figure 3.3). Then, a letter is associated to each average value present in the vector obtained by the PAA algorithm in the following way: the mean values below the lowest breakpoint are mapped with the letter 'a'; all the entries of the vector with values greater than or equal to the first breakpoint and smaller than the second one are mapped to the letter 'b', and so on. Therefore and successively the SAX algorithm assigns a set of letters to each of the vectors previously constructed by the PAA algorithm. In the end of the representation process, we will have the vector represented by one string with the same amount of letters as its dimension.

Figure 3.4 shows a time series with one normalized variable, where six segments were considered for the PAA algorithm. Then, four breakpoints were defined in the Gaussian probability density function, resulting in the partition of possible mean values into five intervals. Subsequently, a letter of the alphabet was associated in an ordered way for each area of values. For example, the average value of the normalized variable in the second segment is above the first breakpoint and below the second one defined in the Gaussian function. So, this segment has been associated with the letter 'b'. Moreover, since the variable mean value

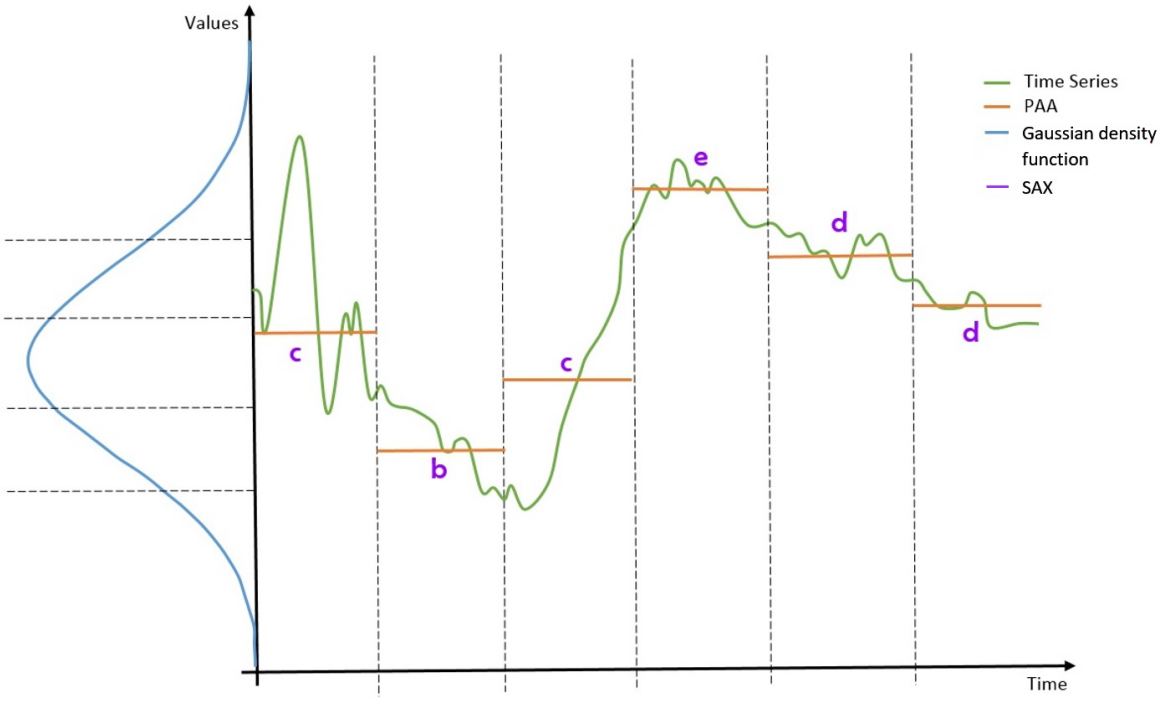

Figure 3.4: Application of SAX algorithm in a univariate time series.

in the fourth segment is above the last breakpoint of the probability density function, this segment was assigned with the letter 'e'. Proceeding successively, in this way the time series represented as a vector of mean values is transformed into a string. In this particular case, the resulting string is 'cbcedd'.

Note that a greater number of partitions considered for the probability density function results into a more rigorous representation of the mean values. However, this also results in a more complex and varied set of possible strings, making the learning of such patterns harder. Moreover, since the data set under study is representative of real behaviors, it is known that noise exits in the data. Therefore, not all the failures follow a strictly equal behavior so an overly detailed representation is not desired. In the end, we note that the SAX algorithm requires the choice of both the number of temporal segments in which the time series is divided (PAA algorithm) and also for the number of letters considered to represent the mean values. The balance between computational cost and loss of behavior specificities should be taken into account.

3.3 Trend Analysis

A study carried out in [11] suggests that an additional phase should be performed after the application of the SAX algorithm. The idea is to associate to each temporal segment, not only the letter representative of the mean value, but also its tendency of growing, decreasing or stability. This new addition of information seeks to approximate the time series representation

to human intuition. In fact, not only the values assumed by the variables are important to interpret the process, but also their behavior over time. So, the objective is to capture the trend in each temporal segment constructed in the PAA algorithm. The attribution of the trend information is made through its association to a straight line. The search for the line that best fits each variable behavior is performed using the least squares method, that is, we intend to obtain the model $y = ax + b$ such that $\sum_{t=1}^{s}(y_t - (ax_t + b))^2$ is minimized, where x denotes the time variation, with $t = 1, \ldots, s$ considering the s existing values registrations of the variable in the segment under transformation, and y represents the value of the variable in each time point t. Note that this phase of the time series representation is made for each segment, similarly to the steps taken in the SAX algorithm. Then, after the line is obtained, its slope is used to assign a second letter to the variable under study in this time segment: 'G' for growth,'D' for decay or 'S' for stable.

It should be noted that with this new representation, the string size obtained by the SAX algorithm has now the double size. This happens because we associate, for each variable and for each time segment, two letters: one letter from the SAX algorithm representative of the mean value and another letter corresponding to the variable trend for that segment. Let us illustrate the process with the following example of a univariate time series (see Figure 3.5). After the application of the SAX algorithm with seven temporal segments and seven letters to represent the mean values, we obtain the sequence 'cegedcc', which is represented in purple color. Then, a straight line, represented by the green color, was adjusted to the variable in each temporal segment. According to the line slope, a second letter was attributed to each temporal segment. This last attribution is represented with a gray color. In the end, the whole time series is represented by the string 'cDeGgDeDdGcDcS'.

Note that this variation to the usual application of the SAX algorithm has advantages, despite the increase both in the complexity of preprocessing of the time series and in the learning process of the classification algorithm. However, this new adaptation can be crucial, as we demonstrate in Figure 3.6. Applying the SAX algorithm in its original form, we would obtain the sequence 'ecced' to represent both time series. However, the behavior is different, mainly in the first segment. So, the sequence obtained from de SAX algorithm isn't enough to distinguish the two time series. If we consider the adaptation presented before, the first time series is now represented by the sequence 'eGcDcGeGdD' whereas the second time series is characterized by the string 'eDcDcGeDdD'.

In conclusion, since the behavior over time is the most important feature to the problem at hand, the variation of the SAX algorithm will be considered to extract information from the time series, in order to allow the implementation of common classifiers.

Let's make a review based on the general vision presented in Figure 3.7. Since we are not able to apply classification methods directly to our multivariate time series, the approach

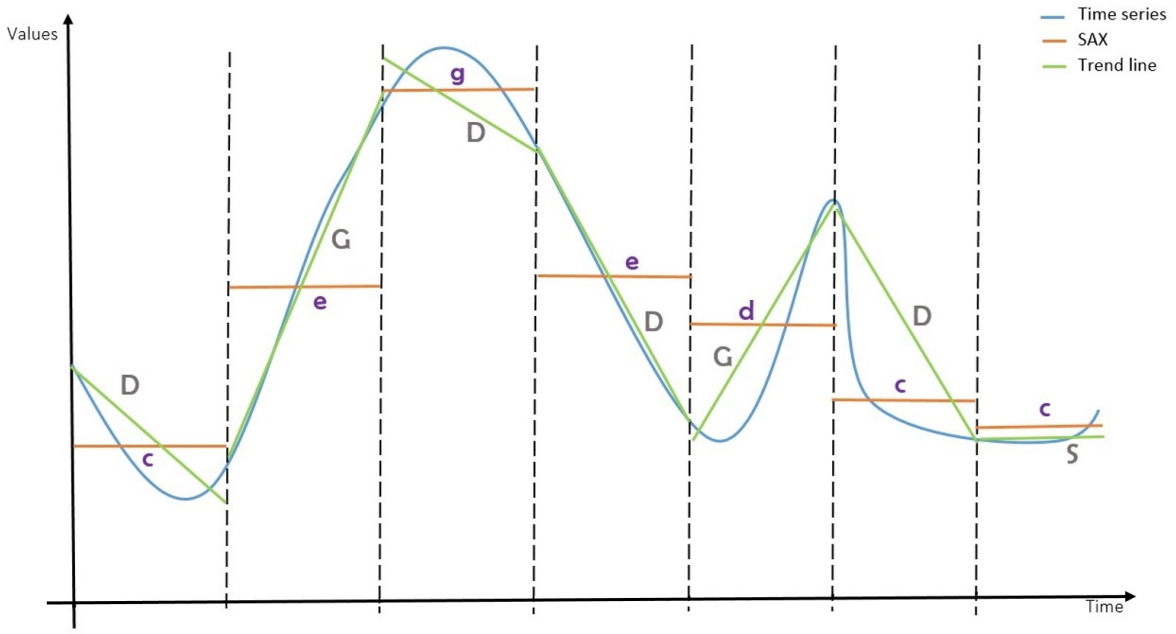

Figure 3.5: Value-Trend approach applied to a univariate time series.

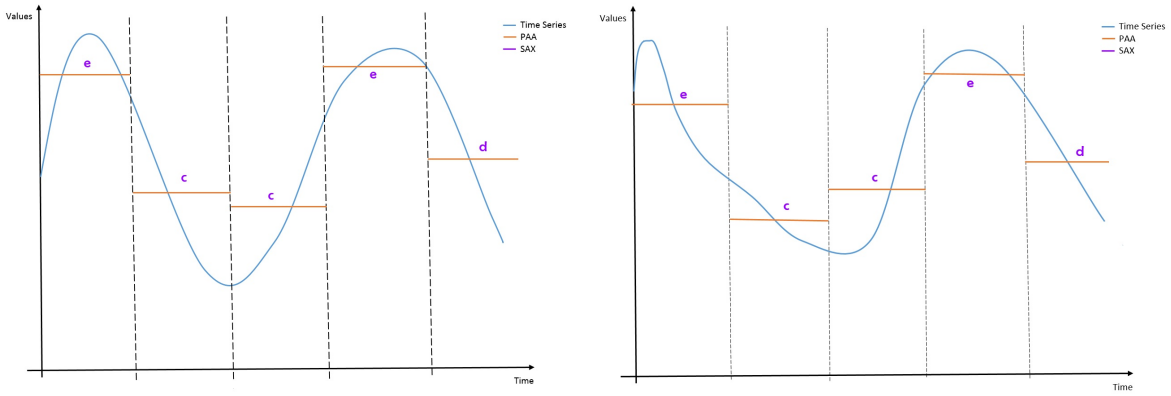

Figure 3.6: Application of SAX algorithm in two univariate time series.

discussed before has the main goal of representing the time series in a simplified way and to allow the implementation of some common methods. This data transformation is made considering each variable individually.

The first step consists in the application of the PAA algorithm, which divides the time series into as much segments as we define. Then, it calculates the mean value for each segment, resulting in a vector of mean values with the same size as the number of segments. After that, the SAX algorithm is applied. It starts by fitting a Gaussian distribution to the value of the time series and partitioning it into as much parts as the letters that we choose to represent the mean values. Then, it attributes a letter to each mean value obtained in the PAA algorithm step, based on the previous Gaussian distribution partition. With the goal of capturing the variable trend in each segment, a straight line is fitted considering the least square method. Then, based on the line slope, another letter is assigned to each segment. The

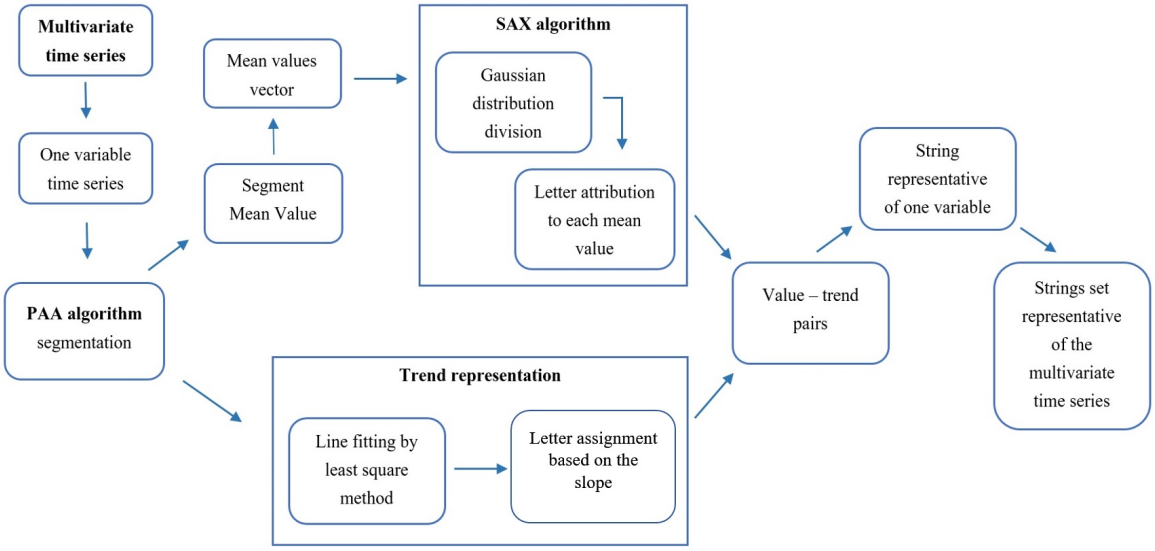

Figure 3.7: Process general vision.

last step consists in joining the two phases of representation for each segment, considering the letter from the SAX algorithm and the one obtained in the trend representation. This process is performed for each segment. In the end, we will get a string with the double length of the considered number of segments. Considering this approach for each variable of the multivariate time series, we will have the original time series converted to as many strings as its dimension.

Figure 3.8 shows the main processing steps starting from a multivariate time series with k variables and m time registrations. As one can see, considering the time series partition into n segments, the final representation of the matrix is a set of k strings with $2n$ length each.

$$\mathbf{TS} = \begin{bmatrix} x_{11} & x_{12} & ... & x_{1k} \\ x_{21} & x_{22} & ... & x_{2k} \\ ... & ... & ... & ... \\ x_{m1} & x_{m2} & ... & x_{mk} \end{bmatrix} \Rightarrow \mathbf{PAA} = \begin{bmatrix} < w_{11}, w_{12}, ..., w_{1n} > \\ < w_{21}, w_{22}, ..., w_{2n} > \\ ... \\ < w_{k1}, w_{k2}, ..., w_{kn} > \end{bmatrix}^\top$$

$$\Downarrow$$

$$\mathbf{SAX} = \begin{bmatrix} < s_{11}, s_{12}, ..., s_{1n} > \\ < s_{21}, s_{22}, ..., s_{2n} > \\ ... \\ < s_{k1}, s_{k2}, ..., s_{kn} > \end{bmatrix}^\top \wedge \mathbf{Trend} = \begin{bmatrix} < t_{11}, t_{12}, ..., t_{1n} > \\ < t_{21}, t_{22}, ..., t_{2n} > \\ ... \\ < t_{k1}, t_{k2}, ..., t_{kn} > \end{bmatrix}^\top$$

$$\Downarrow$$

$$\mathbf{VTA} = \begin{bmatrix} s_{11}t_{11}s_{12}t_{12}...s_{1n}t_{1n} \\ s_{21}t_{21}s_{22}t_{22}...s_{2n}t_{2n} \\ ... \\ s_{k1}t_{k1}s_{k2}t_{k2}...s_{kn}t_{kn} \end{bmatrix}^\top$$

Figure 3.8: Processing steps of the value-trend approach.

3.4 Classification Methods

After the transformation of the time series through the value-trend methodology, it is now possible to apply some common techniques of machine learning to solve the proposed classification problem. Two methods of supervised learning were chosen, and some variants were considered in order to obtain the most appropriate model.

3.4.1 Parameter Selection and Evaluation

The first step of the learning process consists in dividing the data set of labeled observations into three mutually disjoint sets: the training, validation and test sets. Several techniques exist to form these sets with the main goal of avoid overfitting: the construction of a excessively complex model that describes the noise instead of the most interesting patterns. The main steps of the learning process consist in training the model with the training set, that is, fit the model parameters to the data and then evaluate and adjust the actual model with the validation set, to best tune the parameters. Note that the validation set is also used to build the model. After that, the model performance is evaluated with the test set. In the end, outlier identification is performed in the whole data collection with labeled and unlabeled observations, since it is known that some faults were not automatically identified and so, the goal is to identify them in the whole time series collection.

The hold-out technique is the most popular data splitting method for its efficiency and

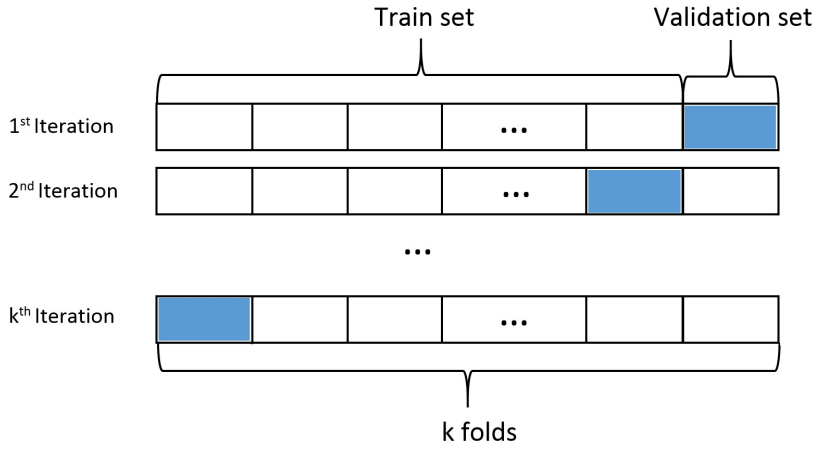

Figure 3.9: k-fold cross validation process.

easiness. It simply divides the whole data set into three sets based on the user choice for each set proportion. One important step of this technique is how to split the data set. Note that it's relevant to obtain sets with the same proportion for each existent class in order to avoid the model preference for classifying one specific class - usually the class with highest proportion in the data.

The k-fold cross validation is another common data splitting technique. The k-fold cross validation consists in partitioning the data set into k equal-sized subsets and using one of such subsets for the validation of the model and the other $k-1$ as the training set (see Figure 3.9). Then, this process is repeated k times, considering each subset as the validation set only once. In this way, it is assured that all observations are used both for training and validation of the model. It remains to choose the value of k. If k equals the number of data set observations, then the model validation is made with one observation and the training process uses all the other observations. This particular case is named leave-one-out cross validation. The choice of k is made in order to minimize evaluation measures. However, the usual number of folds considered in the machine learning community is 5 or 10.

3.4.2 Decision Trees

When trying to understand the appliances faults, we found that the causes approximate to some simple logic rules. For example, consider that there is a order to use hot water but the temperature at the output is not increasing, despite it is bellow the set point. This fault can be traduced into: hot water order $\rightarrow$ temperature not increasing and temperature bellow the set point $\Rightarrow$ fault. So, quickly the decision tree model was the first one making sense to be used for the classification problem as it is also based on simple logic rules.

The decision tree, or more specifically the classification tree, is a non-parametric model commonly used in data mining with the goal of predicting classes. The model construction is based on the learning of simple decision rules inferred from the data and the main steps

consist in dividing the data set into smaller subsets through the incrementally creation of splitting rules. A decision tree model is represented by a tree-shaped diagram where each branch denotes a decision resulted from the node splitting rule. The structure begins with a root node where the first splitting rule divides the original data set into at least two subsets. Then, each node represents a new splitting rule constructed from data set features and resulting in incrementally smaller subsets.

The ID3 algorithm is one of the core decision tree constructors. It begins with the original data set as the root node and, at each iteration, tests every possible splitting rule and calculates some measure of information gain or purity. Then, the splitting rule that maximizes the information gain measure is chosen, producing subsets. After that, the same process is applied for each formed subset, ending when all subset observations have the same class, being the node converted to a leaf and labeled with each subset observations class. It can also happen that there are no more features to be tested and, in this case, the process ends and the majority class of each subset is attributed to the leaf. The classification of new observations is made by making decisions based on the constructed splitting rules, from the root node down to some leaf.

Consider the simplified example of a decision tree model depicted in Figure 3.10, where with few decision rules it was possible to logically characterize three labels: 'CH normal cycle', 'Flame fault' and 'Pump fault'. If we drive throughout the decision tree schema, each label is characterized by a set of decisions. These characteristics allow to understand the label in the same way as the logic followed to justify the labeling when performing the graphical analysis.

Decision trees are simply understandable and capable of quickly dealing with large data sets, being this a determining factor for its choice in this work. However, it should be noted that sometimes they tend to suffer from lack of robustness and can result in over-complex trees that do not generalize the data well. Despite this, decision trees are effective in representing decisions and behaviors able of classify observations.

In order to classify test observations as the different types of faults or normal cycles, the decision tree model was constructed considering three different predictor splitting algorithms, available in MATLAB. The search for the best split in categorical variables is more computationally challenging than for continuous predictors as seen in [25]. The presented algorithms are implemented in MATLAB to work when the classification problem has more than two classes and the goal is to test the splits in a more efficient way. If we transform the multiclass classification problem into a binary one, for example by grouping all the different faults into the label 'fault' and the normal cycles into 'normal', then the software is implemented to use the "exact" algorithm that tests all the possible splits.

The first algorithm is called "pull left by purity". The algorithm tests which is the best split starting by putting all observations in the right node branch that is being constructed.

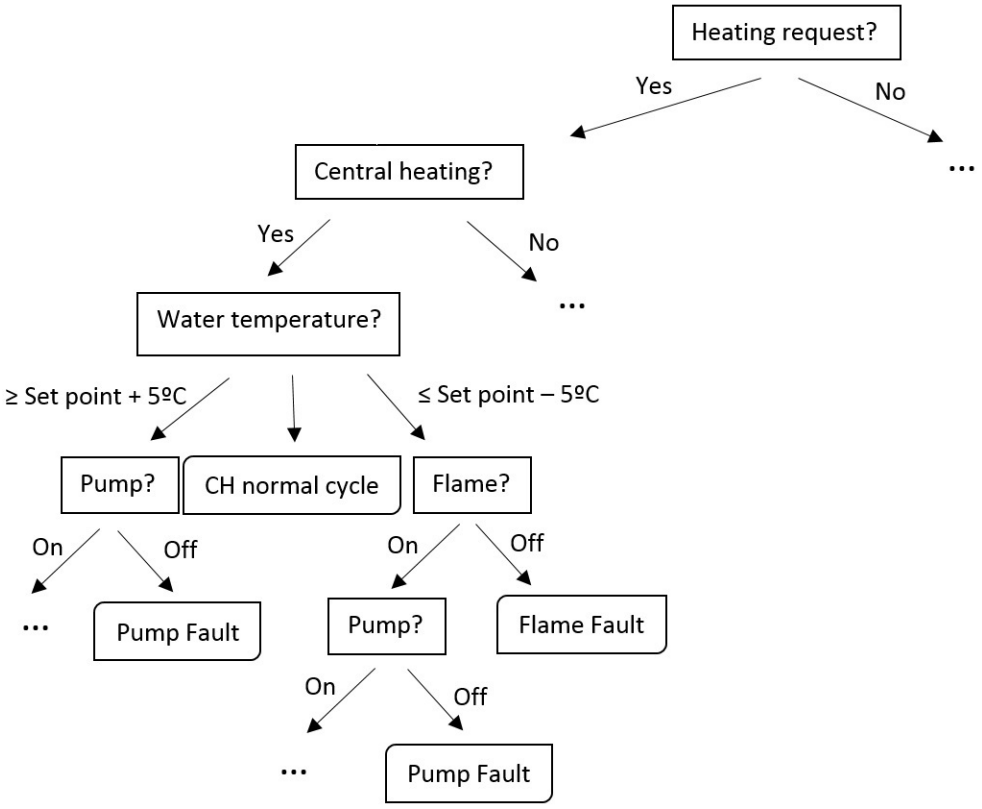

Figure 3.10: Partial decision tree model

Then, for each predictive variable, it calculates which are the strings (values assumed by the predictive variables under study) with highest probability for each of the considered classes. After that, the observations with highest probability strings are moved to the left branch. In this step, the decision rule is associated with the node under study. The algorithm continues sequentially moving the observations with most frequent strings from the right branch of each node to the left one, until the right branch ends with observations of the same string. After the construction of all splits for each predictive variable, the chosen split between all possible studied variables is the one that maximizes the split criterion. Gini's diversity index is the split criterion used by default.

The second algorithm is based on principal components analysis. Named in MATLAB as "principal component-based partitioning", this splitting method assigns a score for each possible string presented in the training set considering each predictive variable separately. The algorithm finds a close-to-optimal binary partition of the possible string set for each predictive variable, through the separating hyperplane search, perpendicular with the first principal component of the weighted covariance matrix of the centered class probability matrix. The score is attributed for each existing string by computing the inner product of the principal component and the vector of class probabilities. The chosen split is the one that maximizes the split criterion.

The third splitting algorithm considered for the decision tree model construction is called "one versus all by class". It starts by considering, for each predictive variable, all existing strings in the forming right node branch. Then, the considered strings are ordered by their probability for each class. Starting with some class, it moves each sequence to the left branch of the node according to its previously calculated probability value starting by the highest. While performing this process, it stores the split criterion for each node. The process is repeated for all considered classes and, in the end, the split that maximizes the information gain measure is the chosen one. Note that the difference between this algorithm and the "pull left by purity" is that, in the former, the competition is made between classes, while in the latter, the predictive variables compete with each other considering all classes at the same time in each step.

It should be noted that other controllable parameters exist in the decision tree model construction, such as the information gain measures and the number of nodes to be considered. This parameters were set automatically by MATLAB. One important step to avoid model overfitting is the decision tree pruning. Reducing the number of nodes and leafs leads to shorter decision rules but leaf impurity increases, since that at each leaf the observations will not correspond all to the same class. However, it is difficult to decide the reasonable number of leafs. The most simple technique consists in cutting the nodes while trying to not affect (significantly) the model accuracy. Therefore, since the possible model construction variations are immense, only a few were tested. The first one was the variation of splitting algorithms presented before. Then, the software capacities were used to optimize the decision tree model parameters such as the maximum number of splits and the minimum of observations needed to form each leaf. All the model parameter choices were made automatically by MATLAB considering the minimization of the cross entropy loss. The main goal was to find the best parameters for the final decision tree model.

3.4.3 k-Nearest Neighbors

Since the data is representative of real-life behaviors, it is not expected to find exactly the same cycles but, instead, similar to those previously observed and labeled. Therefore, the construction of a model based on similarity measures appears to be ideal for solving the presented classification problem. Thus, the second classification method applied was the k-nearest neighbors algorithm, usually denoted as kNN.

Like the decision tree algorithm, the k-nearest neighbors is a non-parametric method, which means that it does not make any assumptions on the underlying data distribution, able of solving both classification and regression problems. This is a commonly used algorithm for its easy interpretation and versatility. kNN is also classified as a lazy algorithm, because all the data is needed during the training phase. So, it requires a costly training phase in terms of time and memory. Yet, this is a simple and effective way of classifying new observations.

The kNN core algorithm consists in searching for the closest training observations around the test observation. Then, the label attributed to the test observation is the one that most appears in the considered closest training observations. The number of closest observations is a user-defined constant, the parameter k, that can vary between one and the training set size. The former is the simplest one, where the label attributed to the test observation is the one from the closest observation in the training set. This case works well when the classes are well separated, otherwise it tends to overfit to the training data. In the latter case, all test observations are labeled with the majority class in the training set, which traduces in a simple but ineffective classifier.

The most common machine learning problems present predictive variables of numeric and static types. So, the commonly used metrics to find the k closest neighbors through the kNN algorithm are Euclidean, Mahalanobis, City block, Minkowski and Chebyshev distances. However, since after the value-trend approach each time series is represented as strings, the main difficulty of applying kNN algorithm to the data set under study is the choice of similarity metrics able of comparing strings.

There exists some measures to evaluate the similarity between strings, with applications in several areas such as spell checking, speech recognition and DNA analysis [26]–[31]. Fuzzy string search, also called approximate string matching, is a group of techniques characterized by their metric: a distance function between words, which is intended to convey a measure of their similarity. Among the most common metrics are the Hamming distance, which expresses the dissimilarity between strings by the number of positions with different characters, equivalent to the minimum number of substitutions required to change one string into another and applicable only to strings of the same length. Therefore, since each time series length is not always the same, it cannot be assured that their representative strings will always have the same number of letters. So, this is not an applicable measure for the studied problem. Another common string similarity measure is the Damerou-Levenshtein, a metric that computes the similarity between strings by calculating the minimum number of operations needed to approximate one string to the other, allowing insertions, deletions, substitutions of a single character or transposition of two adjacent characters. This is not a desirable metric because in this work the order of the letters in the strings has a meaning, corresponding to the value or trend of the time series. So, the transposition of letters can be traduced into a fake and non desirable approximation between strings. Therefore, besides the small variations between the existing string metrics, the choice should be cautious.

The first chosen metric to compute the similarity between two strings, was the Levenshtein distance. Created by Vladimir Levenshtein in 1965 to measure the difference between two strings, this metric considers the minimum number of edits needed to bring one string closer to the other, where the possible edits are insertions, deletions and substitution of letters. In this

metric a higher distance value corresponds to a higher string dissimilarity. The Levenshtein distance is calculated, for two strings s_1 and s_2, as:

$$lev_{s_1,s_2}(i,j) = \begin{cases} max(i,j) & if\ min(i,j) = 0 \\ min & \begin{cases} lev_{s_1,s_2}(i-1,j)+1 \\ lev_{s_1,s_2}(i,j-1)+1 \\ lev_{s_1,s_2}(i-1,j-1)+1_{(s_{1i} \neq s_{2j})} \end{cases} & otherwise \end{cases}$$

Where i is the i-th letter of string s_1 and j is the j-th letter of string s_2.

So, if we consider the strings 'bDaGbGcGdS' and 'aGcGdSdS', the Levenshtein metric would be equal to five because the minimum number of edits needed to bring both strings equal to each other is five, as shown in the following consecutive steps:

aGcGdSdS	bDaGbGcGdS
⇓	*insertion of* b
baGcGdSdS	
⇓	*substitution of* c *for* b
bDaG**b**GdSdS	
⇓	*substitution of* d *for* c
bDaGbG**c**SdS	
⇓	*substitution of* S *for* G
bDaGbGc**G**dS	

The second metric that was considered for the k-nearest neighbors algorithm was based on the longest common subsequence problem. It consists in finding the longest subsequence common to all sequences under consideration in a set of sequences. Therefore, the main goal is to find the biggest sequence of letters in our strings common to all considered (complete) sequences. We illustrate bellow the process with the same strings as before: 'bDaGbGcGdS' and 'aGcGdSdS'. If we consider only the number of common letters between strings, the result is 7 letters (in bold in the first line of the example). Instead, it is required that the sequence of common letters to be consecutive. So, a sequence of 5 letters is the common substring (second line of the example). In the last line, the longest common subsequence method is presented in its original form, where we compare more than two sequences and consider the criteria used in the first line of the example, where the letters sequence do not need to be consecutively equal.

aGcGdSdS ⟷ bD**aG**b**GcGdS**

a**GcGdS**dS ⟷ bDaGb**GcGdS**

a**GcGdSdS** ⟷ bDa**G**b**GcGdS** ⟷ b**GcGdSd**D**cS**

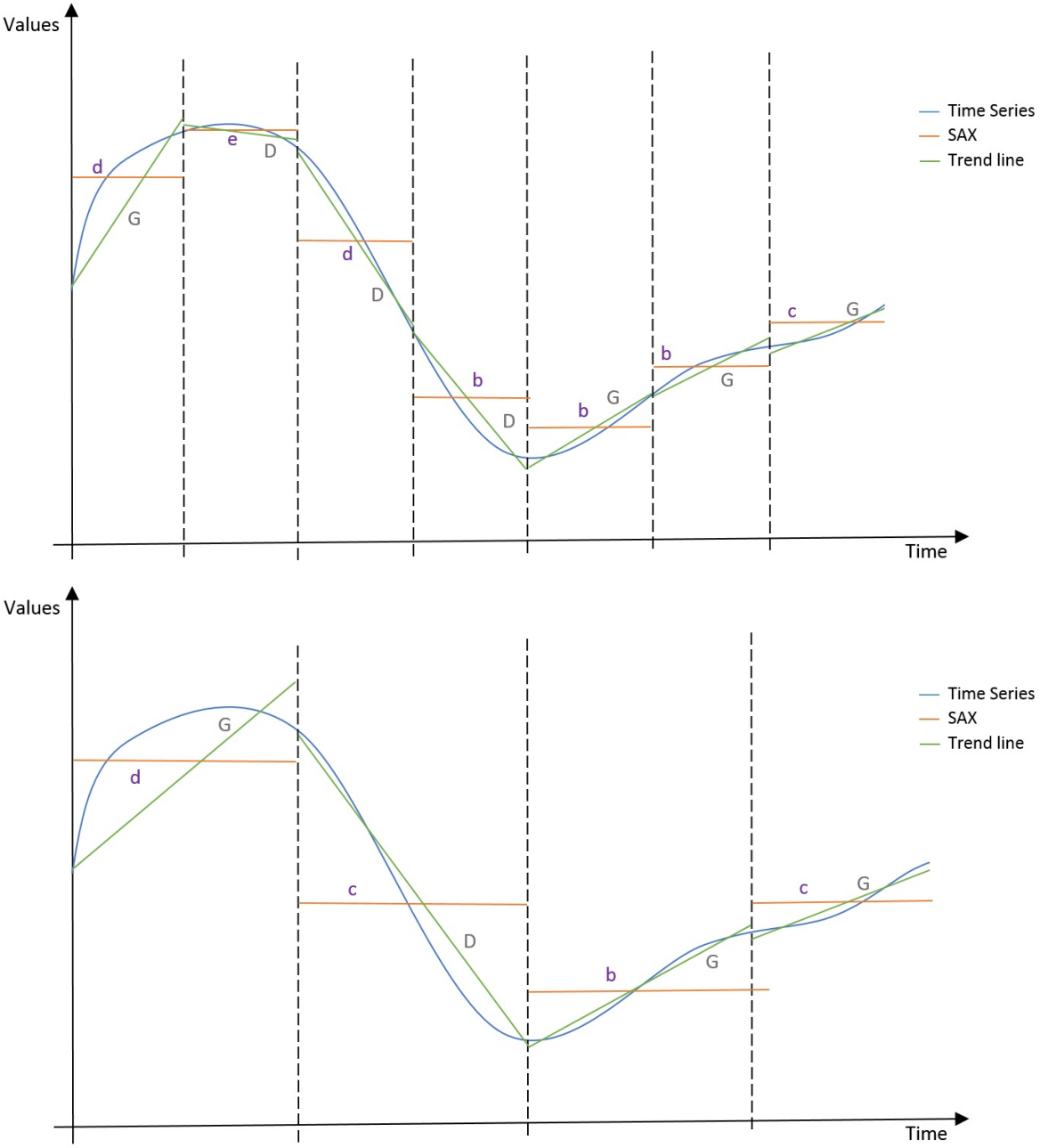

Figure 3.11: PAA segment variation in univariate time series.

Unlike the subsequences method, in the longest common substrings method it is required that the substrings occupy consecutive positions. However, the time series under study have different lengths and so it could result in different sized temporal segments after the application of the PAA algorithm. However two time series can have similar behavior, if different time segments were considered, then the resulting strings are not exactly equal but, instead, they have a set of letters in common. So, it is not intended to have consecutive common substrings but just the highest number of common letters.

Consider the example of Figure 3.11 where the same time series resulted in different strings

('dGeDdDbDbGbGcG' and 'dGcDbGcG') after the application of the value-trend algorithm, due to different number of segments. Since the time series is the same, it is not wanted that the metric between the two strings to traduce high dissimilarity. If the longest common substring method is considered, then the length of the biggest common substring is 4. However, considering the longest common subsequence approach that considers just the number of equal letters, the metric equals to 8. Since we are considering the same time series, obtaining the highest value is more desirable. Therefore, it will be considered that the substrings do not have to occupy consecutive positions. Compared to the Levenshtein distance, the longest common substring metric calculates the minimum number of letters that have only to be removed or inserted in order to the strings become identical.

Besides the metric choice, it is also important to vary the number of neighbors to be considered around the test observations. The appropriate choice of the k value is essential to avoid under or overfitting of the model.

Chapter 4

Implementation and Evaluation

Without big data analytics, companies are blind and deaf, wandering out onto the web like deer on a freeway. - Geoffrey Moore

4.1 Processing the data

Specific dashboards were developed to allow data visualization. One initial step was to analyze the time series supported by plots similar to the one presented in Figure 2.2. With this, a complete knowledge of boiler operation became essential to understand some of the identified faults. This analysis allowed to decide the necessary data time range around faults able of characterizing each specific label. The result was a time range of 30 minutes before the fault identification and 15 minutes after.

Boilers normal operation cycles are not identified by the appliance software and consequently, they are not labeled in the data collection. Yet, some variables let you recognize if there has been any hot water or central heating request. However, at this point it is not clear what defines a correct boiler operation through the behavior of the available predictive variables [32], [33]. As explained before, normal operation cycles are more frequent since the data is collected from boilers working in customers houses. So, there is a need for these labels to be considered in the classification task. Therefore, 100 observations of each HW, CH and boost cycles were manually labeled considering expected behaviors despite there are not defined rules to decide if a cycle is of normal operation or not. Unlike fault occurrences where a fixed time range has been considered, the time range of the normal operation cycles can vary from seconds to minutes. Considering the boiler operation for a bath and for wash some tableware, the time of the heating cycle can vary significantly.

In this first analysis it was also possible to select a subset of variables considered important to explain the faults. Some original variables did not have collected values or any influence on the classes, as seen before in Chapter 2. The result was a subset of 31 predictive variables.

The data was saved in the MONGO database. So, the next step was to save the data from the database to text files, using MATLAB. This step allows a quicker processing of the data and the implementation of machine learning algorithms able to solve the presented problem. The whole collection was not saved, since it concerns several months of about 1500 appliances. Instead, only data from the previous selected subset of variables and the considered time range around the identified faults and normal operation cycles was saved.

The first data processing step was the elimination of repeated lines in order to avoid the problem discussed in Chapter 2, concerning repeated occurrences of the same fault code in a short period of time. Since the data is saved in matrices, with lines representing every millisecond and columns each predictive variable plus the appliance system ID and the time stamp, the solution was simply to join equal lines.

Consider the example of Figure 4.1 concerning two matrices with 4 predictive variables during 60 milliseconds. Note that this is just a part of the saved time range around each fault occurrence. As one can see, the same fault occurred twice in the same appliance with a difference of 40 milliseconds (at $13:30:12.300$ and $13:30:12.700$). Considering the 15 minutes after the first occurrence and the 30 minutes before the second, the number of equal lines would be bigger than in the presented submatrices. With this first processing step of joining equal lines, the result is the third matrix bellow. Notice that now no repeated data exist. Therefore, at the training phase of the models, repeated data will not be considered twice. In this way, the preference from repeated faults is avoided.

It can also happen that, in a short period of time, two or more different faults are identified. If this happens in the same appliance, then equal data is saved since the time range around each fault may intersect totally or just in a subspace of time. However, this data is associated with different labels concerning the distinct faults. Once a column for the label exists, the lines of this matrices differ on the last column. So, the join result is the exact original matrices, since the lines of the matrices are considered as one single joined string and so, the column relative to the label distinguish the strings, and so, the lines of the matrices. Therefore, same data with different labels is not joined, or in other words, data concerning different faults with short time separation is not joined.

The second processing step was the transformation of the state variables into binary values. These variables only assume Yes/No, On/Off, Hot_Water/Central_heating and Okay/Not_okay states and were all coded into 1 and 0, respectively, a necessary step to apply the value-trend approach, because the algorithm is only able to understand the change

Time series 1

$SystemID$	$Time$	$Water\ Temp.$	$CH\ request$	$Flame$	Fan	$Label$
$20 :: dd :: 2cbf :: 213$	$13:30:12.000$	25	No	Off	Off	
$20 :: dd :: 2cbf :: 213$	$13:30:12.100$		Yes			
$20 :: dd :: 2cbf :: 213$	$13:30:12.200$			On	On	
$20 :: dd :: 2cbf :: 213$	$13:30:12.300$	27		Off	On	$f34$
$20 :: dd :: 2cbf :: 213$	$13:30:12.400$	28	No			
$20 :: dd :: 2cbf :: 213$	$13:30:12.500$	27			Off	

Time series 2

$SystemID$	$Time$	$Water\ Temp.$	$CH\ request$	$Flame$	Fan	$Label$
$20 :: dd :: 2cbf :: 213$	$13:30:12.200$			On	On	
$20 :: dd :: 2cbf :: 213$	$13:30:12.300$	27		Off	On	$f34$
$20 :: dd :: 2cbf :: 213$	$13:30:12.400$	28	No			
$20 :: dd :: 2cbf :: 213$	$13:30:12.500$	27			Off	
$20 :: dd :: 2cbf :: 213$	$13:30:12.600$				On	
$20 :: dd :: 2cbf :: 213$	$13:30:12.700$	25			Off	$f34$

Time series 3

$SystemID$	$Time$	$Water\ Temp.$	$CH\ request$	$Flame$	Fan	$Label$
$20 :: dd :: 2cbf :: 213$	$13:30:12.000$	25	No	Off	Off	
$20 :: dd :: 2cbf :: 213$	$13:30:12.100$		Yes			
$20 :: dd :: 2cbf :: 213$	$13:30:12.200$			On	On	
$20 :: dd :: 2cbf :: 213$	$13:30:12.300$	27		Off	On	$f34$
$20 :: dd :: 2cbf :: 213$	$13:30:12.400$	28	No			
$20 :: dd :: 2cbf :: 213$	$13:30:12.500$	27			Off	
$20 :: dd :: 2cbf :: 213$	$13:30:12.600$				On	
$20 :: dd :: 2cbf :: 213$	$13:30:12.700$	25			Off	$f34$

Figure 4.1: Processing repeated data.

$SystemID$	$Time$	$Water\ Temp.$	$CH\ request$	$Flame$	Fan	$Label$
$20::dd::2cbf::213$	$13:30:12.000$	25	0	0	0	
$20::dd::2cbf::213$	$13:30:12.100$		1			
$20::dd::2cbf::213$	$13:30:12.200$			1	1	
$20::dd::2cbf::213$	$13:30:12.300$	27		0	1	$f34$
$20::dd::2cbf::213$	$13:30:12.400$	28	0			
$20::dd::2cbf::213$	$13:30:12.500$	27			0	

Figure 4.2: Encoded state variables of the **Time Series 1**.

of behavior over time with numeric variables. Considering **Time Series 1** of Figure 4.1, this encoding step results in the matrix of Figure 4.2.

The last data processing task was concerned with data continuity. As explained in Chapter 2, matrix empty entries do not always correspond to missing data. A new entry is made only when and in the variables with some change of value. Therefore, data matrices have the same aspect as the ones presented above, full of empty entries.

In order to represent real continuity of values in the matrices, the first approach would be to copy the last saved value until a new entry happens. This would be performed for each column of the matrix, corresponding to each predictive variable, one by one. However, this leads to a fake translation of the boiler operation, because there is one specific fault that is concerned with data loss due to connectivity gateway problems. When the connectivity gateway fails, data is not transmitted. So, copying the data would not allow to identify this fault, since there would be no missing values. Also, a subset of distinct faults are due to different sensors that stop working. For example, if some temperature sensor breaks, there exists a code fault concerning this malfunction. Therefore, the temperature is not measured until the sensor is fixed. By copying the last saved value until a new entry happens in the column concerning this temperature sensor, this would give the idea of a stable temperature over time. However, it could also have changed due to some heat request. With the data copying method, the sensor fault would be impossible to identify, since no missing data would exist.

Through data analysis it was possible to conclude that state variables always register a new entry around every 4 minutes, even if there has been no state change. So, the continuity of values was accomplished by verifying if, for any predictive variable, the last saved value was within less than 4 minutes. If that happens, then the last value of each column is copied until a new value appears. This will allow the identification of the data loss fault, since the lost of data happens during more than 4 minutes. However, in the case of sensor faults it was not possible to avoid the data continuity problem. There is an exception that happens when, for example considering the temperature sensor fault, the variable concerning that temperature

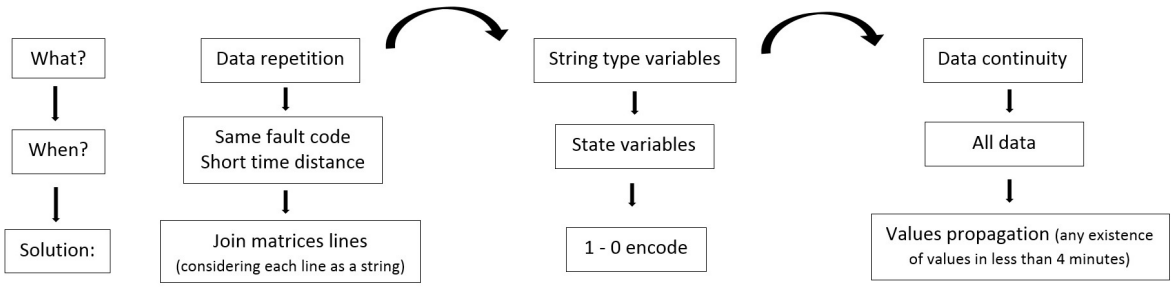

Figure 4.3: Work flow to process the data.

do not have saved values during the considered time range. Therefore, there would not exist values to copy through that column. Therefore, in few cases, sensor faults can be identified by data loss (empty columns) along the time range, together with other data patterns.

4.2 Value-Trend Approach

After processing the data, we apply the value-trend approach followed by the classification methods. All the previous and next steps were performed using MATLAB software. First, the value-trend approach was implemented. Then, the decision tree and k-nearest neighbor were constructed and evaluated.

As seen in Chapter 3 (remind Figure 3.7), the first step of the value-trend approach concerns the time series segmentation: the piecewise aggregate approximation (PAA). Then, the mean value of each variable is calculated and represented by one letter, for each obtained segment. This process is also known as the symbolic aggregate approximation (SAX) algorithm. Main SAX code for MATLAB is available for download[1]. The used function was "sax_modified" implemented in "timeseries2symbol.m" file. The original code receives 5 inputs giving 2 outputs as result. However, this code was adapted to the present problem. Here, 4 inputs are needed: the time series data after the pre-processing steps, in matrix format similar to the examples shown before; number of segments, in order to apply the PAA algorithm; number of letters allowed to represent the time series, defining the Gaussian distribution partition for the SAX algorithm; a binary variable corresponding to the application or not of the Z-score normalization. Since the SAX transformation is made for each predictive variable independently, the first input of the modified algorithm is each data matrix column separately. The output is a string with the same length as the number of defined segments, representative of one variable behavior during the whole time series. In the end, the original data matrix representative of the time series is transformed into a set of strings in the same number as the predictive variables. So, each fault occurrence or normal cy-

[1] `https://cs.gmu.edu/~jessica/sax.htm`

cle corresponds to one matrix line with the same number of columns as the predictive variables.

The output of the SAX algorithm for each predictive variable is a string, representing the mean value of the variable in each segment. The length of this string is equal to the number of considered segments. However, some modifications were made in order to also obtain the trend approximation for each segment. The segments are exactly the same that were considered in the SAX algorithm. The trend representation was made by, in each segment, fitting the variable behavior into one straight line, attending the minimization of the mean square error. For this purpose, "polyfit" MATLAB function was used to obtain the slope of the straight line that best approximates to the data, in each segment. So, the new algorithm output is the string which represents the value-trend representation of each predictive variable. The length of this string is equal to the double of the number of segments: the first and second letters represent the mean value and trend, respectively, of the variable in the first segment; third and fourth letters are the mean value and the trend for the second segment, and so on. In the end, each original time series is transformed into one matrix line of strings with the same number of columns as the predictive variables, plus the label. Therefore, each line corresponds to one occurrence of a fault or normal cycle. This transformation was not applied to the 4 string variables, such as the appliance model. These variables do not change over time and so, do not represent behaviors. However, as explained before, they have interest for this problem.

The choice of the number of segments was based on each time series length. For data relative to more than 10 minutes, the time series is divided into segments of 2 minutes. For time series with length between 2 and 10 minutes, segments of 30 seconds have been formed. In the case of time series with less than 2 minutes, the number of segments is equal to 20 times the time series length in minutes. For example, in a 45 seconds time series (0.75 minutes) the number of segments is $0.75 \times 20 = 15$. So, the whole time series was divided into 15 segments, corresponding each one to 3 seconds. These choices were made in order to best represent the time series behaviors, trying not to catch too much details, but also capture the most important changes.

After processing the data, the set is constituted by two main groups of variables: binary and continuous. Therefore, in the case of the binary variables the Gaussian distribution was partitioned in 2 parts, resulting in 2 possible letters to represent the behavior of these variables. In the case of the continuous variables, the Gaussian distribution was partitioned into 6 parts, and so, 6 letters are available to traduce the behavior of the continuous variables. Note that, as seen in Chapter 3, too many letters increase the problem complexity and a small set of them is not enough to distinguish behaviors.

The original SAX algorithm applies the Z-score normalization to every input data. Since 50% of the predictive variables assume just two states, it does not make sense to apply

normalization to those binary variables. For this reason, one binary input concerning Z-score normalization was added to allow its implementation only when the predictive variables are continuous: the 0 input is used when the variable is binary, meaning that the Z-score normalization is not applied to that variable.

In these conditions, it is now possible to apply machine learning techniques. As discussed in Chapter 3, two methods were selected to solve this problem: decision trees and k-nearest neighbors. The construction of the decision tree model was made using the MATLAB command "fitctree". This algorithm receives as input the training set, the splitting algorithm and a flag pointing that all variables are categorical. The output is a trained model based on the given inputs. Three different algorithms to construct the decision rules were used. All of them are available in MATLAB to be used in "fitctree". After the model is constructed, the MATLAB command "predict" is used to predict the classes of the test set. It receives as inputs the test set, without the response variable, and the fitted model. The output is the predicted class for each test observation.

The second technique was k-nearest neighbors. The model was constructed through the "fitcknn" command, requiring the training set as input, with predictive and response variables separated, a flag pointing that all the predictors are strings and the selected metric. As seen in Chapter 3, two metrics were compared: Levenshtein distance and longest common subsequence. The "predict" command is again used to classify test observations. Given as input the predictive variables of the test set and the obtained model, the output is a class for each observation of the test set.

4.3 Measures of Performance and Results

After de application of the value-trend approach, we are in conditions to apply classification methods. The first one was the decision tree model. For each splitting rule discussed in Chapter 3, one decision tree model was constructed. Also, the MATLAB skills were used to optimize some parameters of each model, such as the minimum number of observations in each leaf node, the maximum number of splits, the split criterion and the number of predictors for split. Figure 4.4 shows the cross validation loss of each decision tree model per splitting rule. The choice of the best values for the parameters is based on the minimization of the cross validation loss. The optimized parameters are shown in Table 4.1.

Split. Rule	Min. Obs. Leaf Nodes	Max. Splits	Entropy Split Crit.	Predict. for Split
PULL LEFT	5	94	Twoing	30
PCP	3	45	Twoing	30
OVA	1	1119	Twoing	30

Table 4.1: Optimized parameters for each decision tree model in the 42-class problem.

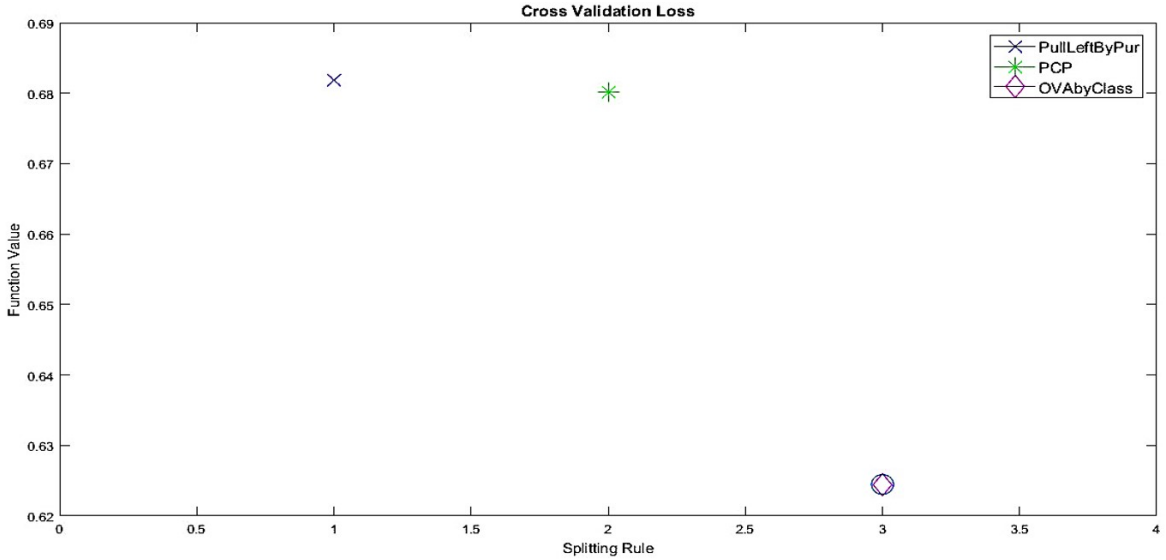

Figure 4.4: Cross validation loss of the three decision tree models considering 42 classes.

As we can see, the "one versus all by class" is the splitting rule that minimizes the cross validation loss. Therefore, a new decision tree model was constructed considering this splitting rule and its optimized parameters. In this phase, the cross validation technique will be considered with 5 folds instead of 10, since there are faults with just one occurrence and so, there will be folds without occurrences of all classes. Some measures of performance, available using the MATLAB command "classperf", are presented in Table 4.2. In outlier identification, normal behaviors are more common than the abnormal ones. So, it is expected that most of the normal behaviors are correctly classified. This leads to a high accuracy, since most of the data is of normal behavior. Accuracy is a measure that expresses the proportion of correct results obtained by the model. Its formula is simply the quotient between the number of correct predictions and the total number of predictions made (usually equal to the number of test observations). However, the goal is to identify the abnormal aspects of the data, so the accuracy does not always best express the performance of a model. Usually, recall is advised. This measure expresses the ability of the method to identify all the relevant items. As it is possible to notice, the accuracy of the obtained decision tree model is low (38.46%).

Classified Observations	99.83%
Accuracy	38.46%

Table 4.2: Performance measures for the optimized decision tree model in the 42-class problem.

The second classification method applied was the kNN and, as discussed in Chapter 3, two metrics were considered to build the models. The metrics are available in MATLAB. The choice of the best model was based on the maximization of the accuracy and the number of folds was been varied from 5 to 10. As we can see in Figure 4.5, exact same results were obtained for both metrics.

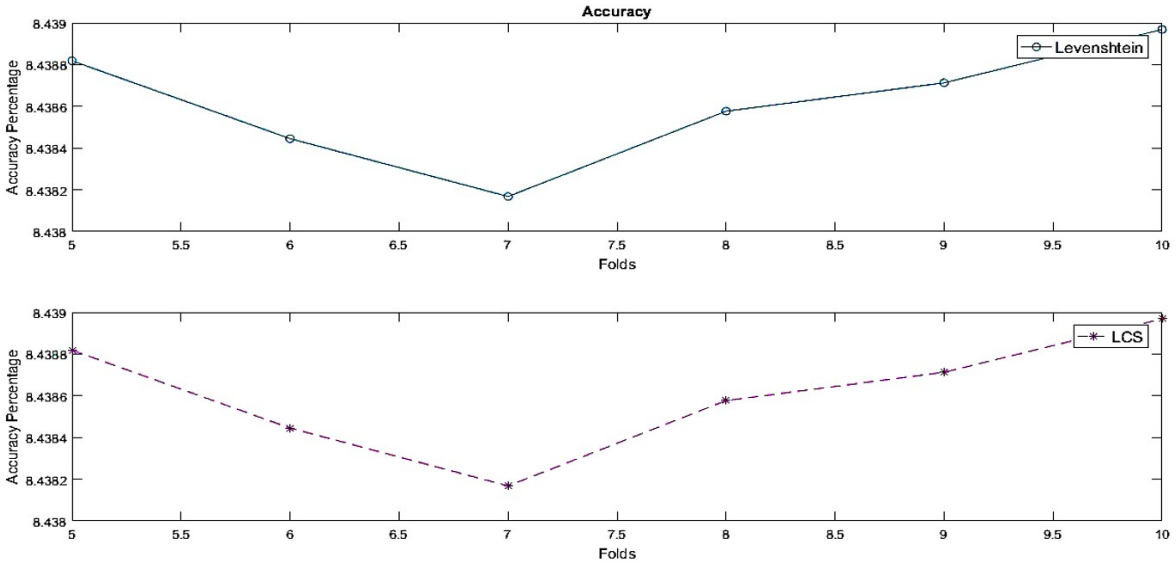

Figure 4.5: Accuracy measure for the kNN model, varying the number of folds from 5 to 10, in the 42-class problem.

When analyzing the prediction results, it was notable to conclude that the kNN model is not suitable for this data set. The predictions were always the "hot water" class for any test observation (corresponding to 2% of well classified observations). Moreover, this prediction happens in both metrics. Accessing the constructed model parameters, we observed that just one neighbor is being considered for the majority vote classification. Forcing the algorithm to consider 2 or more neighbors to classify the test observation, the construction of the kNN model was not possible. The considered metrics compare just one string with other at a time. So, this is a possible reason to consider just one neighbor in the construction of the kNN model. Besides that, the metrics are possibly not adequate to distinguish the cycles. For example, a cycle of 'aGbS' has the same distance from 'aGcS' and 'aGdS'. Suppose that the previous strings represent a temperature variable. Then, 'd' represents a bigger temperature than 'c'. Therefore, an 'a' temperature should be more distant from 'd' than from 'c'. However, the number of substitutions - Levenshtein distance - and the number of equal segments of letters - longest common subsequence - are equal for the first string compared to the second and for the first string compared to the third. So, these metrics are not suitable to distinguish the existent behaviors. Consequently, all cycles are too similar to each other and so, the algorithm is not able to distinguish the classes. The label "hot water" is also the class of the training set first observations. Therefore, this is a possible reason why this class is always attributed to the test observations instead of the others. Since all the cycles have almost the same value for the considered metrics, the first class is the one selected to be attributed to the test observations. Given these results, the kNN model was excluded to solve this problem.

We note that the low accuracy result for the decision tree model is related to the multiclass problem with 42 classes. As discussed before in Chapter 2 (consider Figure 2.3), the data

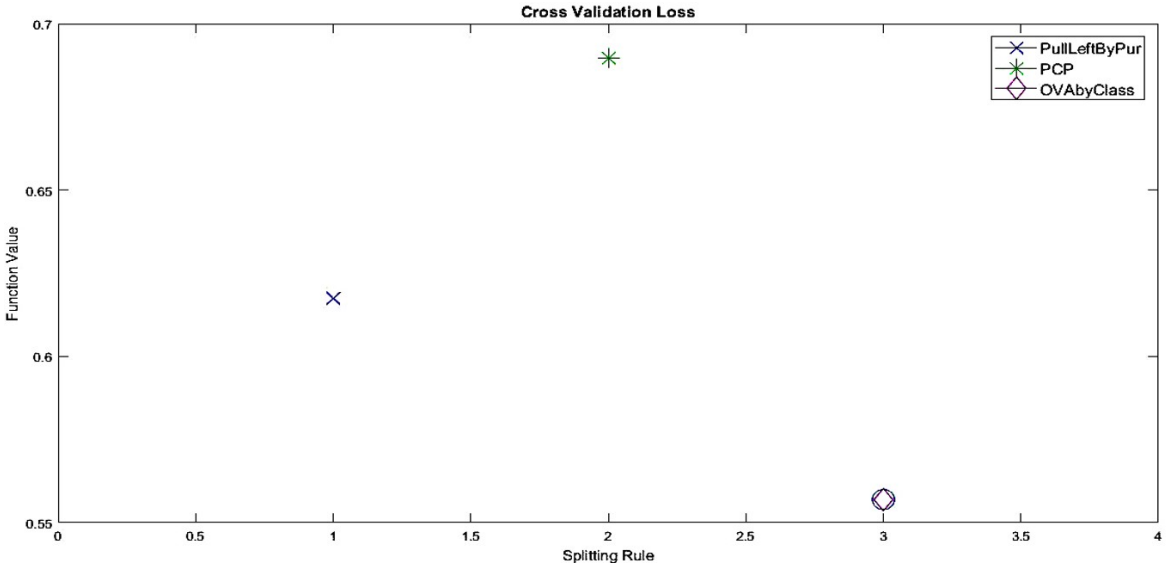

Figure 4.6: Cross validation loss of the considered decision tree models considering 14 classes.

set is not balanced. There are labels with just one occurrence in the total of 1185 classified cycles. Also, the majority of the sets from the cross-validation process do not have at least one occurrence from each label, which has certainly affected the results. Therefore, a new approach was made: the labels were reduced to the classes that present more occurrences than the mean value of occurrences - 11 classes - plus the normal cycles - 3 classes - in a total of 14 classes. The decision tree models were constructed once again considering the 3 discussed splitting rules and making use of the optimization of parameters from MATLAB. Figure 4.6 shows the obtained results and, once again, the "one versus all by class" splitting rule is the one that minimizes the cross validation loss. The optimized parameters are present in Table 4.3.

Split. Rule	Min. Obs. Leaf Nodes	Max. Splits	Entropy Split Crit.	Predict. for Split
PULL LEFT	3	36	Twoing	30
PCP	1	111	Twoing	30
OVA	3	1007	Twoing	30

Table 4.3: Optimized parameters for each decision tree model in the 14-class problem.

Classified Observations	100%
Accuracy	44.30%

Table 4.4: Performance measures for the optimized decision tree model in the 14-class problem.

Then, a decision tree model was constructed considering the "One Versus All by Class" splitting rule and its optimized parameters. The cross validation technique was performed considering 5 folds and some measures were obtained (consider Table 4.4). However the accuracy remains low, the results were better than considering the 42-class problem (Table 4.2). Also, in the 14-class problem, the observations were all classified. In Table 4.5, the

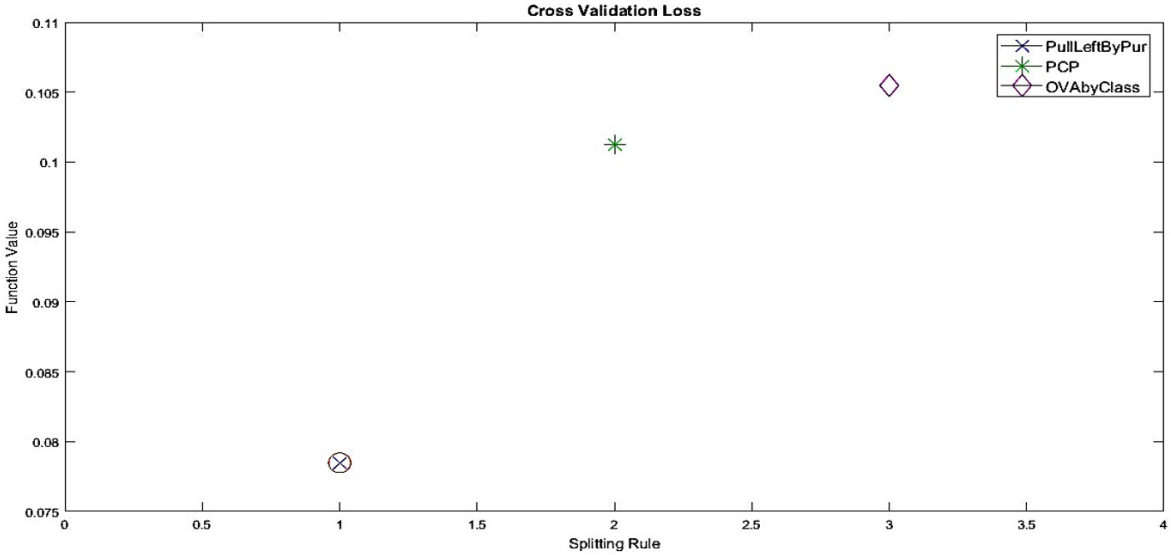

Figure 4.7: Cross validation loss of the considered decision tree models considering 2 classes.

percentage of misclassified observations per class is presented. It was not identified any relation between the percentage of misclassified observations and the number of occurrences per class.

Class	HW	CH	Boost	Fault 1	Fault 2	Fault 3	Fault 4
% Misclassified	52.38	38.80	81.05	71.09	76.67	100	82
Class	Fault 6	Fault 9	Fault 14	Fault 15	Fault 16	Fault 17	Fault 33
% Misclassified	100	90	83.33	100	100	55.33	58.08

Table 4.5: Percentage of misclassification by class, considering the decision tree model in the 14-class problem.

Another aim of this study is to know the lifetime of each boiler model based on the frequency of faults along time. Appliances with more faults, have a shorter life. So, in the following step, the classes were divided into just two classes: fault or normal cycle. This allows to evaluate the appliances with more confidence, since it is not always needed to know which fault occurred. Therefore, a new decision tree model was constructed and optimized, also considering the 3 splitting rules discussed before. The results of the cross validation loss are shown in Figure 4.7 considering the automatic optimization of some parameters. The obtained optimized parameters are presented in Table 4.6. As we can see, considering 2 classes, the "pull left by purity" splitting rule was the one that minimizes the cross validation loss. Therefore, a new decision tree was constructed considering the chosen splitting rule and its optimized parameters along with the technique of cross validation with 5 and 10 folds. Since we are considering just 2 classes, there are now enough observations to apply the cross validation technique with $k = 10$. The obtained measures are presented in Table 4.7. As expected, much better results for the accuracy measure were obtained compared to the previous approaches.

Split. Rule	Min. Obs. Leaf Nodes	Max. Splits	Entropy Split Crit.	Predict. for Split
Pull Left	252	630	GDI	30
PCP	4	200	GDI	30
OVA	175	2	Deviance	30

Table 4.6: Optimized parameters for each decision tree model in the 2-class problem.

As we can see by the value 100% obtained in the "classified observations" measure, the algorithms were both able to classify all the observations. Instead, in the 42-class problem (consider Table 4.2) the decision tree model was only able to classify 99.83% of observations. The specificity was, in both cases of 5 and 10 folds, higher than the sensitivity. Briefly, the sensitivity measures the proportion of observations of the "positive" class that are correctly classified, while the specificity measures the proportion of observations correctly classified as the "negative" class. In this study, the "positive" class corresponds to the normal cycles and the "negative" class to the faults, and so, we can conclude that the algorithm is better at classifying faults than normal cycles. The accuracy reflects how close the classified observations are from the true class, while the precision transmits how consistent the results are when repeating the classification. So, the decision tree model trained with 5 folds have classified the true class more times than the 10 folds decision tree models, while the precision of the results were better in the 10 folds trained model, in other words, the 10 fold decision tree models are more consistent. The main goal of this work is the construction of an algorithm able to identify faults. Therefore, the 5 folds constructed decision tree model is more useful to the company, since is better at identify the faults, which was the main task of this work. The F1-measure reflects the combination of these two last metrics. So, better results were obtained when considering the 5-folds cross validation technique (82.15%), which is expected since this problem suffer from number of observations when considering more than 2 classes.

Measure	5 folds	10 folds
Classified Observations	100%	100%
Sensitivity	79.00%	60.33%
Specificity	95.48%	99.55%
Accuracy	91.31%	89.62 %
Precision	85.56%	98.22%
F1-measure	82.15%	74.75%

Table 4.7: Performance measures for the optimized decision tree model in the 2-class problem.

Chapter 5

Conclusions

When human judgment and big data intersect there are some funny things that happen. - Nate Silver

The main goal of this work was to identify faults in boilers produced by Bosch, through the use of machine learning algorithms. The data was presented in the form of multivariate time series and the faults were classified as outliers. Since some observations were labeled with specific fault codes, the present task quickly became a classification problem.

The main difficulty was to find an algorithm able of classifying time series with predictive variables in the form of strings, continuous and discrete values. Also, processing the data in order to better express the real behaviors of the boilers had became an unexpected difficulty of this work (for example, the representation of the continuity of values in the data matrices).

The final solution was to transform the time series in order to allow the application of common machine learning algorithms. Using the value-trend algorithm, each variable of the time series was coded as one string. In this way, it was possible to express all the different types of predictive variables into a single one (strings). Also, the behaviors previously represented in a set of matrix lines were transformed into a single string (one line matrix). Therefore, each time series became a single matrix of strings, and so, a multivariate classification problem of string type variables.

After that, decision tree and k-nearest neighbors models were constructed. Some variations were considered, like the splitting rules and the measures of similarity, in order to test different model parameters and obtain the best results for the performance measures. As presented in Chapter 4, the decision tree model was the one able to solve the problem.

However, since the results were not satisfactory, a final variation was made. Instead of the 42 labels, the problem was transformed into a binary classification: "normal operation cycle" and "fault" labels were considered. Therefore, the last algorithm is able to identify the faults, however not being capable of distinguish them into each fault code. As seen in Table 4.7, this variation presented much better results.

Considering the whole collection of data, it is known that time series with unidentified faults exist. The result of this study was the identification of these boiler faults. However, it is not possible to evaluate the obtained classifications, since there is no information about the unidentified faults. Therefore, an approximation to the results presented in Chapter 4 is expected for the faults identification in those time series.

Future work includes the prediction of faults in a space time of one week before. Also, it is important to distinguish the faults of the sensors, corresponding to empty matrix lines, from the continuity of values, also correspondent to empty matrix lines before the data processing. A new metric should have been chosen for the k-nearest neighbors model and some techniques to overcome the unbalanced data set can be considered to improve the results of the multi-class problem.

Bibliography

[1] T. V. Phuong, L. X. Hung, S. J. Cho, Y.-K. Lee, and S. Lee, "An anomaly detection algorithm for detecting attacks in wireless sensor networks", 2006.

[2] V. Chatzigiannakis, S. Papavassiliou, M. Grammatikou, and B. Maglaris, "Hierarchical anomaly detection in distributed large-scale sensor networks", 2006.

[3] D. Janakiram, A. M. R. V., and A. P. Kumar, "Outlier detection in wireless sensor networks using bayesian belief networks", 2006.

[4] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos, "Online outlier detection in sensor data using non-parametric models", 2006.

[5] Y. Zhang, N. Meratnia, and P. Havinga, "Outlier detection techniques for wireless sensor networks: A survey", 2010.

[6] A. F. Hassan, H. M. O. Mokhtar, and O. Hegazy, "A heuristic approach for sensor network outlier detection", vol. 1, no. 4, Dec. 2011.

[7] D. Q. Hien, "Time series outlier detection in spacecraft data", Master-Thesis, Technische Universität Darmstadt, Dec. 2014.

[8] L. Wei, N. Kumar, V. Lolla, E. Keogh, S. Lonardi, and C. A. Ratanamahatana, "Assumption-free anomaly detection in time series",

[9] Y. Mao, W. Chen, C. Yixin, and C. Lu, "An integrated data mining approach to real-time clinical monitoring and deterioration warning", 2012.

[10] M. Pimentel, D. Clifton, L. Clifton, and L. Tarassenko, "A review of novelty detection", *Signal Processing*, vol. 99, Elsevier, Ed., pp. 215–249, Jun. 2014.

[11] B. Esmael, A. Arnaout, R. K. Fruhwirth, and G. Thonhauser, "Multivariate time series classification by combining trend-based and value-based approximations", in *Proceedings of the 12th International Conference on Computational Science and Its Applications - Volume Part IV*, Springer-Verlag, 2012, pp. 392–403.

[12] Z. Cui, W. Chen, and Y. Chen, "Multi-scale convolutional neural networks for time series classification", May 2016.

[13] L. Wang, Z. Wang, and S. Liu, "An effective multivariate time series classification approach using echo state network and adaptive differential evolution algorithm", *Expert Systems with Applications*, vol. 43, pp. 237–249, 2016.

[14] S. Kasetty, C. Stafford, G. P. Walker, X. Wang, and E. Keogh, "Real-time classification of streaming sensor data",

[15] S. Smyl and K. Kuber, Eds., *Data Preprocessing and Augmentation for Multiple Short Time Series Forecasting with Recurrent Neural Networks*, presented at the 36th International Symposium on Forecasting (2016), Santander, Jul. 2016.

[16] M. W. Kadous and C. Sammut, “Classification of multivariate time series and structured data using constructive induction”, *Machine Learning*, Feb. 2005.

[17] T. Górecki and M. Luczak, “Multivariate time series classification with parametric derivative dynamic time warping”, *Expert Systems with Applications*, no. 42, pp. 2305–2312, Nov. 2014.

[18] E. "Formisano, F. De Martino, and G. Valente, “"multivariate analysis of fmri time series: Classification and regression of brain responses using machine learning"”, *"Magnetic Resonance Imaging"*, vol. 26, no. 7, pp. 921–934, 2008, "Proceedings of the International School on Magnetic Resonance and Brain Function".

[19] M. W. Kadous, “Learning comprehensible descriptions of multivariate time series”,

[20] (Oct. 2016). Sax-vsm, [Online]. Available: `https://jmotif.github.io/sax-vsm_site`.

[21] J. Lin, E. Keogh, L. Wei, and S. Lonardi, “Experiencing SAX: A novel symbolic representation of time series”, *Data Mining and Knowledge Discovery*, vol. 15, pp. 107–144, 2 Oct. 2007.

[22] B. Alsallakh, M. Bögl, T. Gschwandtner, S. Miksch, B. Esmael, A. Arnaout, G. Thonhauser, and P. Zöllner, “A visual analytics approach to segmenting and labelling multivariate time series data”, Vienna University of Technology, TDE Thonhauser Data Engineering GmbH, University of Leoben, EuroVis Workshop on Visual Analytics, 2014.

[23] Q. Wang and V. Megalooikonomou, “A dimensionality reduction technique for efficient time series similarity analysis”, Mar. 2008.

[24] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, “A symbolic representation of time series, with implications for streaming algorithms”, Jun. 2013.

[25] (). Splitting categorical predictors, [Online]. Available: `https://www.mathworks.com/help/stats/splitting-categorical-predictors-for-multiclass-classification.html`.

[26] G. Navarro, “A guided tour to approximate string matching”,

[27] P. Christen, “A comparison of personal name matching: Techniques and pratical issues”, Sep. 2006.

[28] W. W. Cohen, P. Ravikumar, and S. E. Fienberg, “A comparison of string distance metrics for name-matching tasks”,

[29] R. Thonangi, “Classifying categorical data”, Master of Science by Research (Computer Sciences), International Institute of Information Technology, Dec. 2005.

[30] B. Knoll, “Text prediction and classification using string matching”,

[31] D. Coppersmith, S. J. Hong, and J. R. Hosking, “Partitioning nominal attributes in decision trees”, *Data Mining and Knowledge Discovery*, vol. 3, pp. 197–217, 2 Jan. 1999.

[32] S. Erkel, “Top 79: Tt domain knowledge for data analysis”, May 2015.

[33] N. Ramakrishnan and S. Erkel, “Top79: Root cause analysis and predictive maintenance for bosch thermotechnology”, May 2016.